

NASA Contractor Report-187544

11-a1

7N-82

2856

P-187

**Advanced Information Processing System:
Design and Validation Knowledgebase**

**Richard E. Harper
Linda S. Alger
Jaynarayan H. Lala**

**THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139**

**Contract NAS1-18565
September 1991**



National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

(NASA-CR-187544) ADVANCED
INFORMATION PROCESSING SYSTEM:
DESIGN AND VALIDATION KNOWLEDGEBASE
Final Report (Draper (Charles
Stark) Lab.) 187 p

N94-71800

Unclass

29/82 0002856

NASA Contractor Report-187544

Advanced Information Processing System: Design and Validation Knowledgebase

**Richard E. Harper
Linda S. Alger
Jaynarayan H. Lala**

**THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139**

**Contract NAS1-18565
September 1991**



**National Aeronautics and
Space Administration**

**Langley Research Center
Hampton, Virginia 23665-5225**

TABLE OF CONTENTS

| Title | Page |
|---|-------------|
| List of Illustrations..... | vii |
| List of Tables..... | ix |
| 1.0 INTRODUCTION | 1-1 |
| 1.1 Need for Validated Architectures | 1-1 |
| 1.2 Design for Validation Methodology | 1-4 |
| 1.3 AIPS Design and Validation Knowledgebase..... | 1-7 |
| 1.3.1 Architecture Knowledgebase..... | 1-7 |
| 1.3.2 Performability Knowledgebase..... | 1-7 |
| 1.3.3 Formal Proofs..... | 1-8 |
| 1.3.4 Other Design and Validation Knowledgebase Components | 1-8 |
| 2.0 ARCHITECTURE KNOWLEDGEBASE | 2-1 |
| 2.1 Introduction | 2-1 |
| 2.2 AIPS Knowledgebase..... | 2-1 |
| 2.2.1 Purpose..... | 2-2 |
| 2.2.2 Definitions..... | 2-3 |
| 2.2.3 Approach | 2-3 |
| 2.2.3.1 Overview | 2-3 |
| 2.2.3.2 Overview-Knowledgebase for Centralized Architectures..... | 2-4 |
| 2.2.3.3 Overview- Knowledgebase for Distributed Architectures | 2-4 |
| 2.3 Requirements..... | 2-6 |
| 2.4 Attributes | 2-7 |
| 2.5 Rules, Specifications, and Guidelines | 2-9 |
| 2.6 Directed Graphs..... | 2-11 |
| 2.7 Conclusions and Recommendations..... | 2-17 |
| 2.7.1 Conclusions | 2-17 |
| 2.7.2 Recommendations..... | 2-17 |
| Appendix 2.A Requirements..... | 2-20 |
| Appendix 2.B Attributes | 2-26 |
| Appendix 2.C FTP Specification..... | 2-50 |
| Appendix 2.D I/O Net Specification | 2-59 |
| Appendix 2.E IC Net Specification..... | 2-61 |
| Appendix 2.F System Services Software Specification..... | 2-64 |
| Appendix 2.G Guidelines | 2-72 |
| 3.0 FORMAL VERIFICATION OF INTERACTIVE CONSISTENCY..... | 3-1 |
| 3.1 Introduction | 3-1 |
| 3.2 Approach | 3-1 |
| 3.3 Outline of Development..... | 3-2 |
| 3.4 Assumptions | 3-3 |
| 3.5 Informal High-level Specifications of a Byzantine Resilient Interactive | 3-4 |
| Consistency Algorithm..... | 3-4 |
| 3.6 Formal High-level Specification of Algorithm OM..... | 3-6 |
| 3.7 Conclusions and Recommendations..... | 3-11 |
| 4.0 AIPS FOR ALS ANALYTICAL MODELING..... | 4-1 |

| | | |
|---------|---|------|
| 4.1 | Introduction..... | 4-1 |
| 4.2 | Software Specifications..... | 4-1 |
| 4.2.1 | FTP Markov Model | 4-2 |
| 4.2.1.1 | VLSI FTP Architecture | 4-2 |
| 4.2.1.2 | FTP Failure Rates | 4-3 |
| 4.2.1.3 | FTP Reliability Modeling Assumptions | 4-5 |
| 4.2.1.4 | AIPS FTP Markov Model..... | 4-6 |
| 4.2.1.5 | Modeling Approach | 4-9 |
| 4.2.2 | IC Network Markov Model | 4-11 |
| 4.2.2.1 | Assumptions | 4-14 |
| 4.2.2.2 | IC Network Markov Models..... | 4-15 |
| 4.2.2.3 | Pad Model..... | 4-15 |
| 4.2.2.4 | Launch Model..... | 4-16 |
| 4.2.2.5 | IC Bus Model..... | 4-16 |
| 4.2.2.6 | Modeling Approach | 4-21 |
| 4.2.2.7 | Combinatorial Formulation..... | 4-22 |
| 4.2.3 | Reliability Analysis Results | 4-26 |
| 4.2.4 | Reliability Analysis Conclusions | 4-28 |
| 4.3 | Requirements and Performance Modeling..... | 4-29 |
| 4.3.1 | Performance Analysis Approach..... | 4-29 |
| 4.3.1.1 | Performance Analysis Approach | 4-29 |
| 4.3.1.2 | Architecture Synthesis Outputs | 4-29 |
| 4.3.2 | Advanced Launch System (ALS) Functions | 4-30 |
| 4.3.2.1 | Processing Specifications | 4-30 |
| 4.3.2.2 | I/O and Interfunction Communication Specifications | 4-30 |
| 4.3.3 | AIPS Performance Model | 4-31 |
| 4.3.3.1 | Virtual Architecture Model | 4-31 |
| 4.3.3.2 | Schedule Model..... | 4-32 |
| 4.3.3.3 | Frame Model..... | 4-32 |
| 4.3.3.4 | ADAS Model..... | 4-33 |
| 4.3.4 | Requirements and Performance Modeling Results | 4-39 |
| 4.3.5 | Requirements and Performance Modeling Recommendations | 4-40 |
| 5.0 | EMPIRICAL TEST AND EVALUATION..... | 5-1 |
| 5.1 | Introduction | 5-1 |
| 5.2 | AIPS Architectural Parameters..... | 5-2 |
| 5.3 | Performance and Reliability Metrics..... | 5-6 |
| 5.4 | FTP Empirical Knowledgebase..... | 5-10 |
| 5.4.1 | FTP Architecture and H/W and S/W Implementation Technology | 5-10 |
| 5.4.2 | FTP Performance Data..... | 5-13 |
| 5.4.3 | FTP Reliability Parameters | 5-14 |
| 5.5 | Input/Output Network Empirical Knowledgebase..... | 5-16 |
| 5.5.1 | I/O Network Architecture and Hardware and Software Implementation Technology..... | 5-16 |
| 5.5.2 | I/O Network Performance Data | 5-20 |
| 5.5.3 | I/O Network Reliability Parameters | 5-22 |
| 5.6 | Inter-Computer Network Empirical Knowledgebase..... | 5-26 |
| 5.6.1 | IC Network Architecture and H/W and S/W Implementation Technology..... | 5-26 |
| 5.6.2 | IC Communication Performance Data..... | 5-29 |
| 5.7 | Impact of Advanced Technology Insertions..... | 5-29 |
| 5.7.1 | Software Technology Insertion - Compilers | 5-30 |
| 5.7.2 | Hardware Technology Insertion - Microprocessors..... | 5-32 |
| 5.8 | Conclusions | 5-33 |

| | |
|---------------------------------------|-----|
| 6.0 SUMMARY AND CONCLUSIONS..... | 6-1 |
| 6.1 Architecture Knowledgebase..... | 6-1 |
| 6.2 Performability Knowledgebase..... | 6-1 |
| 6.3 Formal Proofs..... | 6-3 |
| 7.0 REFERENCES..... | 7-1 |

LIST OF ILLUSTRATIONS

| Figure | Title | Page |
|--------|--|------|
| 1-1. | Conventional Avionics Design Methodology..... | 1-3 |
| 1-2. | AIPS Design for Validation Methodology | 1-5 |
| 2-1. | Graded Redundancy Attribute..... | 2-14 |
| 2-2. | Top-Level Relationship Graph | 2-16 |
| 3-1. | Physical Arrangement of Fault Containment Regions..... | 3-3 |
| 3-2. | Detail of a Single Fault Containment Region | 3-4 |
| 3-3. | Interactive Consistency Data Flow..... | 3-5 |
| 3-4. | Timing Diagram for Interactive Consistency | 3-6 |
| 4-1. | Quadruplex Fault Tolerant Processor Architecture..... | 4-3 |
| 4-2. | FTP Markov Model..... | 4-7 |
| 4-3. | Probability of Fault Masking Capability after 1 Week Unattended on Pad | 4-10 |
| 4-4. | Probability of Launch Loss at End of 10-Minute Boost..... | 4-11 |
| 4-5. | Unreconfigurable Quaduply Redundant Bus with Two Quad FTPs..... | 4-12 |
| 4-6. | AIPS Engineering Model Configuration | 4-13 |
| 4-7. | Pad Model of the InterComputer Network Transmitters and Receivers..... | 4-17 |
| 4-8. | Launch Model of the InterComputer Network Transmitters and Receivers..... | 4-19 |
| 4-9. | Model of the InterComputer Network: Active Failure Mode..... | 4-20 |
| 4-10. | Inter-FTP Communication Paths for CMC=3..... | 4-23 |
| 4-11. | Possible Communication Loss Configurations for 3 FTPs..... | 4-25 |
| 4-12. | AIPS Virtual Architecture..... | 4-31 |
| 4-13. | AIPS FTP Virtual Architecture..... | 4-32 |
| 4-14. | Schedule Module..... | 4-33 |
| 4-15. | FTP Frame Activity..... | 4-34 |
| 4-16. | ADAS Model of Scheduler | 4-35 |
| 4-17. | CP Subgraphs..... | 4-36 |
| 4-18. | IOP Subgraphs (1)..... | 4-37 |
| 4-19. | IOP Subgraphs (2)..... | 4-38 |
| 4-20. | I/O Network Subgraph..... | 4-39 |
| 5-1. | AIPS Hardware Building Blocks..... | 5-3 |
| 5-2. | Fault Tolerant Processor Architecture: Functional View (One Channel) | 5-12 |
| 5-3. | Ada Run Time System & Scheduler Overheads for FTP Model 1..... | 5-14 |
| 5-4. | Ada Run Time System & Scheduler Overheads for FTP Model 2..... | 5-15 |
| 5-5. | Redundancy Management Overhead - No Fault Conditions..... | 5-17 |
| 5-6. | Redundancy Management Overhead - Data Exchange Fault..... | 5-18 |
| 5-7. | Redundancy Management Overhead - Unsynchronized Channel | 5-19 |
| 5-8. | Centralized AIPS Configuration | 5-21 |
| 5-9. | I/O Communications Management Overheads..... | 5-23 |
| 5-10. | I/O Request Processing (10 Hz Task)..... | 5-24 |
| 5-11. | I/O Redundancy Management Overheads..... | 5-24 |
| 5-12. | I/O Redundancy Management: Failed Leaf Node | 5-25 |
| 5-13. | I/O Redundancy Management: Single Chain Grow..... | 5-25 |
| 5-14. | AIPS Engineering Model | 5-27 |
| 5-15. | IC Communication Overhead..... | 5-30 |
| 5-16. | Dhrystone (Compiler) Benchmark Results | 5-31 |
| 5-17. | Whetstone (Compiler) Benchmark Results | 5-32 |

LIST OF TABLES

| Table | Title | Page |
|-------|--|------|
| 4-1 | FTP Device Failure Rates ($\pi_e=3.0$, $\pi_d=0.5$) | 4-4 |
| 4-2. | Definition of Symbols | 4-8 |
| 4-3. | Pad Failure Rates..... | 4-8 |
| 4-4. | Launch Failure Rates | 4-9 |
| 4-5. | Pad Model Symbol Definition and Numerical Values | 4-18 |
| 4-6. | Launch Model Symbol Definition and Numerical Values | 4-18 |
| 4-7. | Analytical Results for Quad FTP..... | 4-27 |
| 4-8. | Analytical Results for Triplex FTP..... | 4-27 |
| 4-9. | Analytical Results for Quad IC Bus (CMC=3) | 4-28 |
| 5-1. | AIPS Architectural Parameters | 5-3 |
| 5-2. | Performance Metrics..... | 5-7 |
| 5-3. | Reliability Metrics..... | 5-9 |

1111

ADVANCED INFORMATION PROCESSING SYSTEM: DESIGN AND VALIDATION KNOWLEDGEBASE

1.0 INTRODUCTION

The overall objective of the Advanced Information Processing System (AIPS) program is to develop the knowledgebase which will allow achievement of validated fault tolerant distributed computer system architectures, suitable for a broad range of applications, including those which have a failure probability requirement as low as 10^{-9} at 10 hours. This knowledgebase consists of an architecture knowledgebase containing a set of AIPS architecture attributes, design rules and specifications, and guidelines that are traceable to top level mission requirements. It also contains a performability knowledgebase comprising quantifiable performability data, organized as analytical and empirical relationships between performance metrics, reliability metrics, and architectural parameters. Other knowledgebase components not described in this report include the AIPS hardware and software building block requirements and specifications, several simulations which have been written to validate various aspects of the AIPS, the results of empirical evaluations of the AIPS engineering model, and a technology survey for the Advanced Launch System application. The specific quantitative and qualitative design objectives of AIPS are discussed later in this report.

1.1 Need for Validated Architectures

A validated architecture is defined to be an architectural concept that when implemented in hardware and software will meet various mission requirements such as reliability, throughput, transport lag, cost, weight, volume, power, etc. Computer system reliability and performance are of paramount importance for real time safety- and/or mission-critical applications. Existing methodologies for validating these attributes of a computer system architecture, before its implementation in hardware and software, are either nonexistent or ad hoc in nature. Invariably, the reliability and performance of the actual system fall short of the mission requirements. This entails costly revisions to the architecture, hardware, and software which means that the mission cost and schedule goals are also not met. Furthermore, for safety-critical systems such as commercial transport fly-by-wire applications the computer system reliability can not be validated with existing ad hoc validation methods. For example, the state-of-the-art in validation cannot show to a determined certification authority that a given computer system meets the ultra-low failure probability requirement of 10^{-10} per hour for 10 hours that is required in such safety-critical applications. An architecture knowledgebase and a validation methodology are needed that will allow the achievement of validated fault tolerant computer system architectures for real time safety-critical and/or mission-critical applications. This has been the overall objective of the AIPS program.

Even for the Advanced Launch System which has only a moderate mission reliability requirement (failure probability of 10^{-5}), the current ad hoc avionics design and validation techniques are neither adequate for validation nor cost effective in producing a validated avionics suite. The cost of avionics, both in absolute terms and as a fraction of the total cost of the vehicle, continue to increase. This is in spite of the decreasing cost of a unit of computation as measured by dollars per MIP throughput or dollars per Mbyte storage. Some of the increase can be attributed to the increased functionality of avionics which requires higher throughput and more memory and software. However, with increased functionality comes an increasing dependence on the correct operation of avionics. This requires an added architectural dimension, viz. fault tolerance, that is not present in conventional computer system architectures. Along with fault tolerance comes hardware and software redundancy, management of redundancy, detection and isolation of faults, reallocation of resources and many other complexities. Fault tolerance also affects system performance. For example, the overheads of redundancy management reduce the throughput available to the applications programs and various error checking layers generally impede the flow of data through the computer system resulting in higher data latencies. Since the target applications of these fault tolerant avionics require high performance real time operations, the performance effects of fault tolerance are just as important as the reliability aspects. The validation of these avionics architectures is consequently becoming a complex multi-dimensional problem. One must validate the fault-free system performance, i.e., show that the system meets various throughput, timing, and other performance requirements when there are no faults in the system. It is also necessary to validate the fault tolerance attributes, i.e., show that the system can detect and isolate faults and recover from them in a timely fashion and with the requisite probability. The synergism between these two aspects must also be validated, i.e., show that the system performance during fault handling and after reconfiguration is acceptable. It is this multi-dimensional validation that is driving the cost of avionics skyward even as the unit costs of computation are declining dramatically.

The traditional avionics design methodology also adds to the overall cost of avionics as well as to the schedule slips because it is no longer adequate to address the complex validation issues. In the traditional design process, the shortcomings of the computer architecture are not discovered until the Full Scale Engineering Development (FSED) phase of the design cycle. Sometimes, only after test and evaluation of the FSED article is it possible to say whether the design meets the mission objectives and requirements. For example, redundancy management issues which not only impact the system's ability to tolerate faults but also affect the throughput performance and data latencies are typically left undefined until the FSED phase. The unacceptable performance penalties and inadequate fault coverage that result from incorrect redundancy management designs are discovered in the implementation phase. To correct any deficiencies in the architecture at this point requires a loop back to the architecture synthesis phase and a very time consuming and expensive iteration through the design cycle, as shown in Figure 1-1.

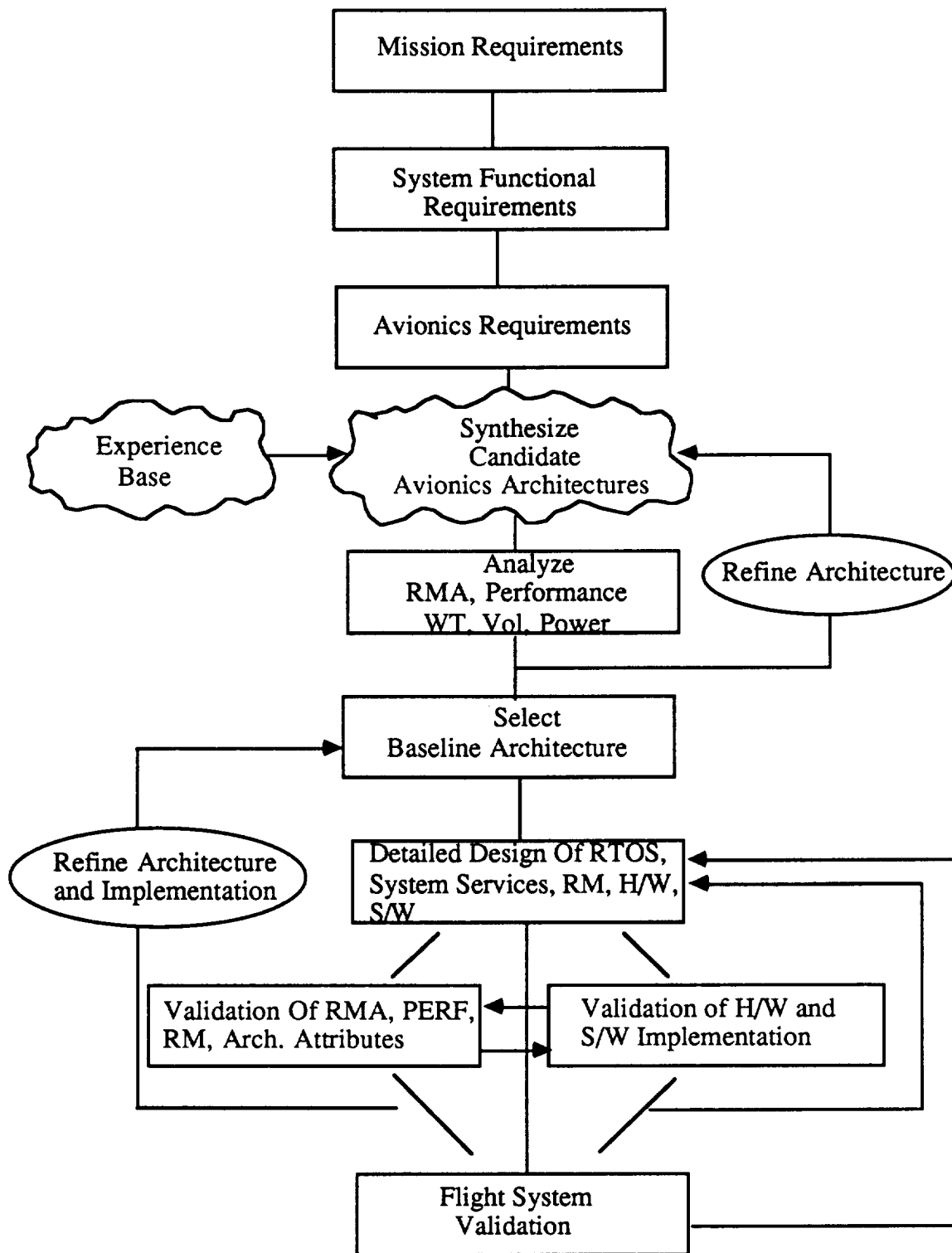


Figure 1-1. Conventional Avionics Design Methodology

Another problem commonly discovered in the FSED phase is the incorrect partitioning of functions between hardware and software. Typically, a lot of time consuming functions that can be more efficiently implemented in hardware are discovered to be in software that consumes too much throughput and is not fast enough. There is a very simple explanation of why this problem crops up repeatedly in real time fault tolerant computer systems. The knowledgebase required to make many design decisions affecting performance, fault tolerance and redundancy management does not exist or is not available to designers. As such, a number of design issues are postponed to be resolved at some later point in the design cycle. As the detailed hardware design progresses, the hardware implementation options for system functions start to close out leaving the system software the burden of solving and implementing all the unresolved design issues. There is also the illusion of leaving different design options open by not committing them to hardware. When the inadequate performance or fault tolerance is discovered in the FSED phase, it is very expensive to go back and transfer functions from software into hardware. It should be pointed out here that analytical modeling of the architecture before the start of the detailed hardware and software design will help correct these problems if and only if the architecture is well understood so that it can be modeled with high fidelity. If, for example, the redundancy management design has not been finalized one can not estimate the fault coverage and other parameters that are required for a high fidelity model.

1.2 Design for Validation Methodology

A new design for validation methodology has been developed as part of the AIPS program to reduce the cost of developing and validating fault tolerant computer system architectures. This design methodology is depicted in Figure 1-2. Although the overall flow of the design cycle starting from mission requirements and ending with a flight system appears somewhat similar to the conventional avionics design methodology there are several key differences that will be highlighted here.

The first important difference is the manner in which an architecture is synthesized to meet the avionics requirements. In the traditional methodology the architecture synthesis task is a subjective art form that depends on the creativity, biases and past experiences of the designers. In the design for validation methodology, a set of functional requirements is derived from the mission requirements and translated into avionics requirements. These avionics requirements are then mapped into prevalidated hardware and software building blocks using a knowledgebase and future technology projections. Validation of the AIPS building blocks and generation of the knowledgebase and technology projections are goals of the AIPS program. The validation is being performed using a combination of requirements acquisition, design for Byzantine resilience, mathematical proofs, analytical models, and empirical test and evaluation. The architecture knowledgebase allows the designer to synthesize the architecture in accordance with rules and guidelines such that the fundamental principles of fault tolerance are adhered to and rationale for each design decision is related to overall mission requirements. The building block knowledgebase

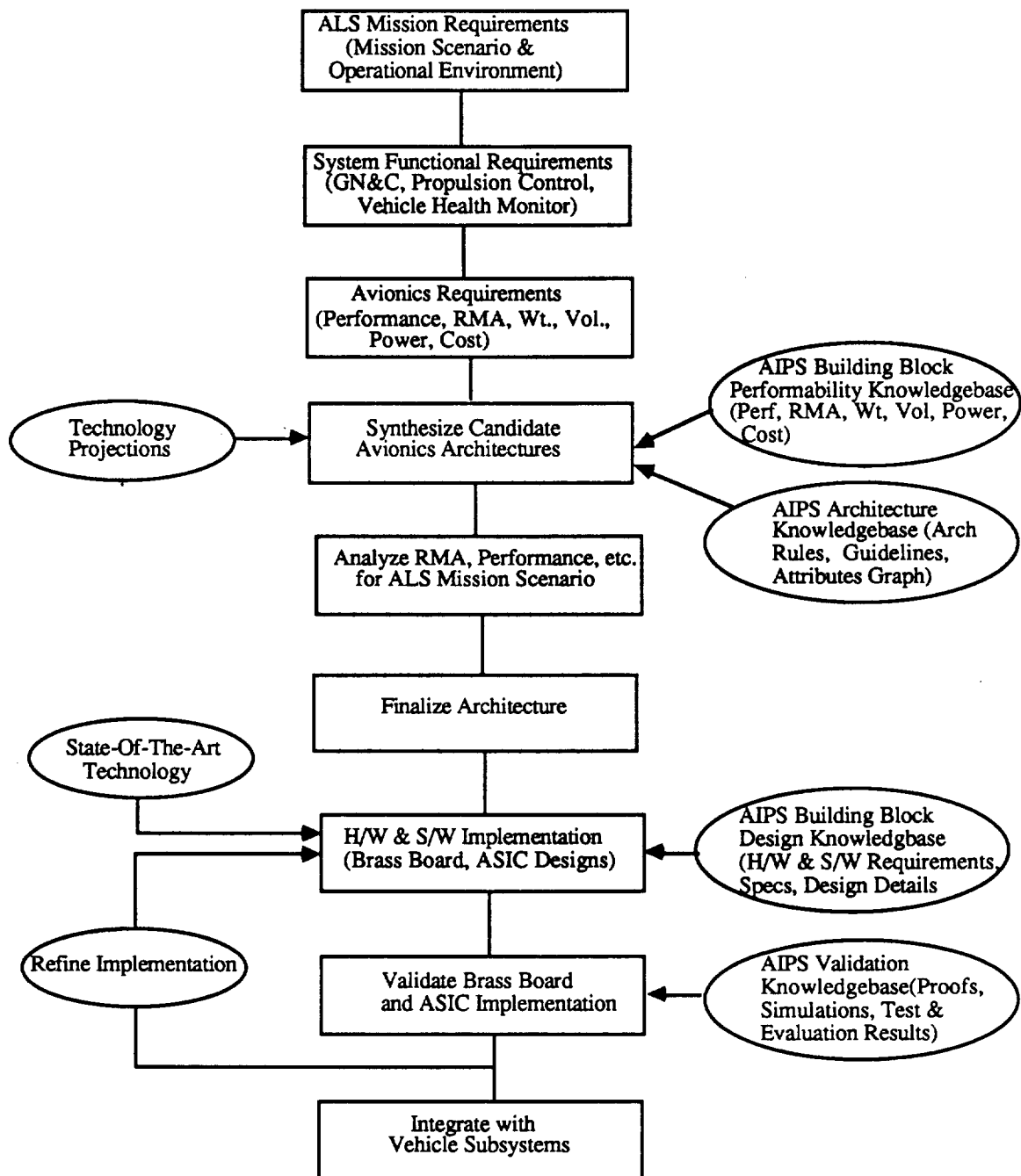


Figure 1-2. AIPS Design for Validation Methodology

provides the designer with a detailed characterization of the performability and other important parameters of each building block. These characterizations, in conjunction with advanced technology projections, can be used to project the expected performability of building blocks implemented in state-of-the-art hardware and software technology. The AIPS hardware and software building blocks have been designed such that their major attributes such as Byzantine resilient fault tolerance, simplex programming model, reconfigurability, rigorous separation of redundancy management and applications software, etc. are not dependent on any specific technology of implementation. Furthermore, the system services are implemented such that their overheads become smaller as the processor and communication speeds increase. Therefore, the building blocks do not become obsolete with technology advancements. Their performance increases in direct proportion to improvements in processing and communication speeds.

This design methodology makes the architecture synthesis task much less of an art and personal judgement and provides a solid foundation of knowledgebase on which to base design decisions. A second important difference in the two design methodologies is the validation of the architectural concept and its realization in hardware and software. In the conventional design methodology, validation of the architectural characteristics such as redundancy management, reliability, maintainability, performance, etc. is done in parallel with the validation of the specific hardware and software implementation of the design. When it turns out that the system cannot meet the mission requirements, it is not clear whether the basic architectural concepts are flawed or the specific hardware and software implementation of the concept has some shortcomings.

The design for validation methodology decouples the validation of the architecture design from the validation of a specific hardware and software implementation of that design. The combination of architectural rules and guidelines, the prevalidated building blocks knowledgebase and the analytical models of the performability of the synthesized architecture assure a validated architecture. As stated previously, a validated architecture is defined to be an architectural concept that when implemented in hardware and software will meet various mission requirements such as reliability, throughput, transport lag, cost, weight, volume, power, etc.

The detailed hardware and software designs can commence at this point, i.e., after an architectural concept has been synthesized. Once the brassboard has been fabricated and programmed, test and evaluation of the brassboard along with the usual hardware and software verification tools can be used to ascertain compliance of implementation to architectural concepts. Only those aspects of the implementation that were changed due to technology upgrade need to be revalidated. For example, if the processor module in the AIPS Fault Tolerant Processor is upgraded but the data exchange gate array does not change then only the processor module needs to be revalidated. If problems are encountered in this phase then they point to the implementation and not the architecture itself. For example, if the redundant processors in the FTP fail to synchronize then a likely

problem may be a lack of clock determinism in the new processor module rather than a basic flaw in the synchronization algorithms or the Fault Tolerant Clock. (Clock deterministic hardware behavior is one of the AIPS architectural rules). The hardware and software design errors in the brassboard can be corrected relatively inexpensively and much faster than changing the architectural concept. Once the implementation errors have been found and corrected, a validated flight system can be fabricated.

1.3 AIPS Design and Validation Knowledgebase

The design for validation methodology, as shown in Figure 1-2, needs a substantial body of knowledge that we collectively call the architecture design and validation knowledgebase. Such a knowledgebase has been created for the AIPS architecture over the past few years. This report presents a part of that knowledgebase. References to other parts of the knowledgebase which are not covered in this report are given in Section 1.3.4

1.3.1 Architecture Knowledgebase

The architecture knowledgebase consists of a set of system architecture attributes, design rules and specifications, and guidelines that are traceable to various top level mission requirements. As shown in [16], many missions have common top-level requirements. A novel methodology, based on directed graphs, has been used to organize the many interrelationships between these components of the AIPS knowledgebase. This organization provides a framework for many uses of the AIPS architecture knowledgebase. System designers who are not necessarily experts in fault tolerance can use it to configure the AIPS hardware and software building blocks to meet specific mission requirements. Certification authorities can use it to validate implementations of AIPS building blocks. The knowledgebase can also be used as a pedagogical tool to explain the AIPS philosophy and architectural concepts. The methodology and the architecture knowledgebase are discussed in Section 2.

1.3.2 Performability Knowledgebase

To configure AIPS building blocks to meet specific application requirements, it is necessary to characterize performability, i.e., performance and reliability, of building blocks and of ensembles of building blocks through fundamental architectural parameters. Such a database would eventually consist of all quantifiable knowledge about the architecture that affects its performability. It is organized as analytical and empirical relationships between three major domains: performance metrics, reliability metrics and architectural parameters. The metrics and the AIPS architectural parameters are described in Section 5 which also discusses the empirical relationships between these three domains using the results obtained on the AIPS engineering model. The empirical data has been collected on both the hardware and the software building blocks of AIPS.

The requirement of extremely low system failure rates for the AIPS applications (typically 10^{-6} to 10^{-10} per hour) precludes computer reliability validation exclusively by means of experimental evaluation. Therefore, a set of analytical models has been developed to characterize reliability and availability of the AIPS hardware building blocks. The models are discussed in Section 4 in the context of the Advanced Launch System mission requirements. The models are, however, general enough so that by changing a few parameters one can predict the reliability and availability for other mission scenarios also. A performance modeling approach to characterize component utilization, data latencies and other performance metrics of AIPS is also described in Section 4.

The analytical modeling and empirical characterization of the AIPS building blocks complement each other. Test and evaluation on the engineering model have been used to verify model assumptions, determine unknown parameters and increase overall confidence, and hence claims of validation, in the system.

1.3.3 Formal Proofs

Formal proofs can play a very important role in validating certain aspects of ultrareliable computer systems. The AIPS building blocks use the results of the formal distributed systems theory, generally known as the Byzantine Generals Problem, to provide the ultrahigh reliability necessary in safety-critical applications. This attribute of AIPS facilitates the use of formal logic to prove certain fault tolerance characteristics of the architecture. Section 3 discusses this approach and its application to the implementation of data consistency and validity in the AIPS Fault Tolerant Processor. It should be noted here that because such proofs can replace the very expensive, tedious and time consuming Failure Modes and Effects Analysis the cost of validation can be reduced substantially for any application that can afford the required hardware redundancy. We believe that this is a strong motivation to design fault tolerant systems such that they lend themselves to formal verification and analytical modeling techniques even if the reliability requirements are only moderately high such as for mission critical applications.

1.3.4 Other Design and Validation Knowledgebase Components

In parallel with this report, several other documents have been produced that contain major components of the AIPS design and validation knowledgebase. These are as follows.

The AIPS software building block requirements and specifications are described in Local System Services [1], Input/Output System Services [2], Input/Output Network Management Software [3], and InterComputer Communication Services [4]. The Fault Tolerant Processor data exchange hardware requirements and specifications are described in [1]. The circuit switched node which is the core of the AIPS I/O and intercomputer

networks is described in [2]. The InterComputer Interface Sequencer (ICIS) is described in [3].

Several simulations have also been written to validate various algorithmic aspects of AIPS. These include the InterComputer Network Contention Algorithm [5], the IC Network Data Source Congruency algorithms [6], and the IC Network Distributed Growth algorithm [4].

The AIPS engineering model which consists of three triplex FTPs and a simplex processor networked together by a 3-layered Inter-Computer network, a 15-node I/O network, and the System Services software (about 100,000 lines of Ada source code) has been operational at the Charles Stark Draper Laboratory for some time. The detailed hardware logic diagrams for the hardware building blocks and the source code listings for the software building blocks have been provided to NASA Langley Research Center previously. The engineering model is now under test and evaluation using a hardware pin-level fault injector. The results of this effort will be published at a later date.

A technology survey was undertaken to forecast the technology projections for the Advanced Launch System application [7]. These forecasts were used to predict the performability of the AIPS building blocks in the ALS time frame. These are summarized in the ALS Architecture Synthesis report [8].

Finally, Section 6 summarizes the state of the AIPS design and validation knowledgebase and recommends which areas should be pursued further to complete this body of knowledge.

2.0 ARCHITECTURE KNOWLEDGEBASE

2.1 Introduction

The goal of the Advanced Information Processing System (AIPS) program is to develop the "Knowledgebase" needed to assemble distributed fault-tolerant system architectures, and reduce the cost and effort that must be dedicated to the development of each individual system by eliminating the need to repeat the development of common elements over and over again. Products of the program are system architecture concepts, a set of building blocks, and design and evaluation tools that can be used together to implement information system architectures for a broad range of NASA missions.

In order to ensure that the AIPS architectural concepts are valid and practical, and that the building blocks are appropriate for implementing the concepts for this broad range of missions, it is necessary to state the functional and operational requirements that must be met by the information processing systems. Therefore one of the first tasks in the knowledgebase development must be to establish a baseline set of functional and operational requirements and objectives that systems assembled using the AIPS architectural concepts and building blocks must meet. These requirements and objectives are discussed in Section 2.3. The attributes obtained from these requirements and objectives are discussed in Section 2.4. Rules and guidelines associated with them are discussed in Section 2.5.

These requirements, attributes, rules and guidelines constitute much of the contents of the AIPS Knowledgebase. A methodology for capturing and expressing this knowledge for both centralized and distributed architectures as well as an overview of the AIPS Knowledgebase are discussed in Section 2.2. Part of the methodology created involves representing the relationships between these requirements, attributes, rules and guidelines in the form of directed graphs. These interactions are discussed in Section 2.6.

Conclusions and recommendations for the AIPS Knowledgebase are provided in Section 7.0. Appendices A to G list detailed requirements, attributes and specifications for AIPS.

2.2 AIPS Knowledgebase

The overall goal of the AIPS program is to develop a "Knowledgebase" which will allow achievement of validated fault tolerant computer system architectures, suitable for a broad range of applications. The knowledgebase comprises the AIPS Architecture Knowledgebase, the AIPS Performability Knowledgebase, Formal Proofs, and other components such as detailed software and hardware building block engineering requirements, specifications and detailed logic diagrams, simulation and reliability models, fault injection results, technology projections, and a common mode failure protection strategy. The development of AIPS Architecture Knowledgebase, consisting of the architecture guidelines and specifications and their relationship to mission requirements and AIPS attributes, will be the focus

of this document. The other components of the knowledgebase are mentioned here to provide a perspective on the total AIPS Knowledgebase.

Key elements of the Architecture Guidelines and Specifications knowledgebase are high-level mission requirements, system architecture attributes, design rules and specifications, guidelines, and a methodology for clearly relating these elements to each other.

The knowledgebase for the AIPS is large and involves many interrelationships among requirements, attributes, rules, specifications and guidelines. These interrelationships may be either intuitive, quantitative, or formally defined. In our methodology these relationships are depicted in a directed graph format to allow a designer unacquainted with the details of fault tolerance technology to understand how an AIPS system must be organized and operated. Viewed from a different perspective, the graphical structure allows a validation and verification authority to examine an AIPS implementation and determine whether it meets the requirements of the applications. The set of requirements, attributes, rules, specifications, guidelines, and their interrelationships thus provides a framework for structuring and interconnecting AIPS building blocks into a computational system which can be shown to satisfy the mission requirements.

2.2.1 Purpose

The primary purpose of the knowledgebase is to provide an organizational framework for designing and understanding the AIPS. Since April 1984, various documents on the AIPS program were published. These documents specified hardware, software, and system design decisions, their rationale, and the requirements and design specifications. The requirements and design specifications were published as formal documents [18-27]. However, since the main objective of the AIPS program has been to explore uncharted territory in the area of distributed fault-tolerant systems, the program has had a very strong research and development content. The goal has been to demonstrate certain novel concepts in a laboratory environment with prototype hardware and software, as opposed to developing a flight qualified system. Consequently, not all the design specifications, requirements, and their associated rationale are formally documented. They exist as informal internal CSDL memoranda authored by designers and implementers of the system for use by other members of the design group.

To reduce the cost of developing and deploying validated fault tolerant systems based on the AIPS concept, an organizational framework, to be known as the AIPS Knowledgebase, is being created. The main idea is to gather all the AIPS information that is currently in various places and produce a methodology which will be able to relay the information to the user. This methodology is intended to serve several purposes. It provides an organizational framework for understanding the AIPS concepts and their motivation, serving as a pedagogy for the theory behind the AIPS. It is also intended to provide a methodology to structure the AIPS design process for those who may not be experts in fault tolerant systems. It is intended to provide traceability of the AIPS design

specifications to requirements and vice-versa, thus enhancing the ease of validation of the eventual AIPS instantiation. Finally, it is intended to clarify issues surrounding fault tolerance-related myths and possible controversy relating to the AIPS approach to fault tolerance.

2.2.2 Definitions

Some of the key words that will be referenced throughout this document will now be defined.

Requirement: A requirement is a statement of an AIPS design objective. A requirement may be either quantitative or qualitative.

Attribute: An attribute is an unambiguous statement of a characteristic of the AIPS. Attributes can be quantitative, qualitative or logical.

Rule: A rule is a principle which must be followed to assure that a higher-level AIPS attribute holds.

Specification: A specification is an aggregate of rules which describe all relevant characteristics of an AIPS component. The specification is intended to be sufficiently detailed to allow one "unskilled in the art" to construct the component. The specifications contain sufficiently rigid and detailed rules so that they preclude the implementer (who may not be an expert in fault tolerance) from introducing single point failures or from violating AIPS concepts that might affect the system performance or reliability in a detrimental fashion.

Guideline: A guideline is a statement of policy or philosophy based on Draper experience, along with a statement of the effects of deviating from the guideline. Guidelines do not contain enough design detail to translate the AIPS concepts into a hardware and software realization. For that to happen, it is also necessary to document a detailed set of design specifications. Guidelines are intuitively motivated and justified primarily through experience or quantitative analysis.

2.2.3 Approach

2.2.3.1 Overview

Some of the AIPS' applications may require only a centralized computer architecture, while others may require distributed computer architectures to meet such requirements as throughput, graceful degradation, physical dispersion and function migration.

With the above in mind, an approach for the development of the AIPS Knowledgebase was developed. The AIPS architecture supports fault-tolerant system

configurations which are either centralized or distributed. Building blocks for centralized systems are also the basic building blocks for the distributed configurations. The AIPS concept is one of appending communication capabilities to centralized building blocks so as to achieve an efficient interconnection of modularized elements. The distributed system is thus an aggregation of centralized elements which are interconnected by a fault-tolerant intercomputer communications network.

2.2.3.2 Overview - Knowledgebase for Centralized Architectures

The approach used to develop the AIPS Knowledgebase involved first examining the AIPS *centralized* architecture. The centralized AIPS architecture consists of both hardware and software building blocks. The hardware building blocks are a Fault Tolerant Processor (FTP), an Input/Output Network, and FTP-Network Interfaces (IOSs). The software building blocks are Local System Services and Input/Output System Services.

Capturing the knowledge for the centralized architectures occurred during an earlier phase of the AIPS program [14]. The architectural guidelines and specifications for centralized versions of the AIPS architecture, such as a single fault-tolerant processor (FTP) or a FTP with a dedicated I/O Network, are well understood though not necessarily exhaustively documented. A major portion of the effort during this earlier phase was the development of this documentation in a formalized manner consisting of the creation of the knowledgebase for centralized architectures. The approach used to develop this knowledgebase involved examining the AIPS design requirements and objectives, articulating the attributes and relationships, aggregating the architecture design rules and specifications, and stating the architecture guidelines.

The requirements and objectives were obtained from references [16, 17]. From these requirements and objectives, a set of attributes was derived. Specifications and guidelines were then compiled. These rules and guidelines were compiled after discussions with Draper personnel as well as an extensive literature search on AIPS. The major documents that were consulted are listed as references [1, 2, 3, 14].

Additional references are listed in Section 7. After the information on the AIPS requirements and objectives, attributes, rules, specifications, and guidelines was compiled, the description of the centralized AIPS architecture was performed. This description began with the development of logical relationships between the AIPS requirements, attributes, and the architectural rules, specifications and guidelines. These logical relationships are depicted in the form of directed graphs, to be discussed in Section 2.6.

2.2.3.3 Overview - Knowledgebase for Distributed Architectures

After capturing the knowledgebase for centralized architectures, it was necessary to expand the knowledgebase to include the rules, specifications, and guidelines relating to *distributed* fault-tolerant information processing system architectures. The computational

core of a distributed information processing system consists of a set of fault-tolerant, Fault Tolerant Processors (FTPs), a communication mechanism for sharing information between FTPs and a mechanism that provides each FTP access to external devices such as sensors, actuators, displays, and other subsystems that are outside the computational core. The core information processing requirements for a broad range of missions can be met by architectures that are constructed with a common set of building blocks. These hardware building blocks can be identified as various FTPs, an Inter-FTP communication mechanism made up of links and nodes (IC Network), and external communication mechanisms (I/O Networks). A fault-tolerant mass memory, and a fault-tolerant power distribution system are also hardware building blocks for distributed architectures. The system software is another AIPS building block. It provides the traditional services necessary in a real-time computer such as task scheduling and dispatching, communication with sensors and actuators, etc. The software also supplies those services necessary in a distributed system such as inter-function communication across processing sites, management of local and distributed redundancy, management of networks, and migration of functions between processing sites.

Capturing the knowledge for the distributed architectures was performed in the current phase of the AIPS program [14]. The approach used to develop this knowledgebase is very similar to that for the centralized architecture.

The two documents that were referenced for the AIPS design requirements and objectives for centralized architectures [16, 17] were again examined. This time the requirements and objectives for distributed architectures, such as Function Distribution, were obtained. In addition to examining these documents, Draper personnel were consulted for their expertise regarding the AIPS program. These consultations resulted in additional AIPS requirements and objectives.

From these requirements and objectives, a set of attributes was derived. For each of the attributes, architecture rules, specifications and guidelines were then compiled. These attributes, rules and guidelines were compiled after discussions with Draper personnel as well as an extensive literature search on AIPS. However, the information on distributed architectures has not yet been fully documented. Most of the information obtained for the distributed architectures was therefore obtained through discussions with experienced Draper personnel.

The information compiled on the AIPS requirements and objectives, attributes, rules, specifications, and guidelines for distributed architectures was then combined with the information obtained from the centralized architectures. From this large collection of information, logical relationships between the AIPS requirements, attributes, and the architectural rules, specifications and guidelines were created and represented in the form of directed graphs.

2.3 Requirements

The scope of the requirements is limited to the information processing, intercomputer communication and input/output interface functions. The requirements include subjective design objectives, quantifiable performance measures, and selected functional characteristics.

The AIPS architecture may be evaluated with respect to both qualitative (subjective) and quantitative (objective) measures. Qualitative evaluation is carried out by examining architectural specifications, operational philosophy, redundancy management techniques, etc. Among some of these qualitative parameters are adaptability, modularity and physical dispersion. The quantitative evaluation results from analytic modeling, simulation, emulation, and test and measurements of actual system. The quantitative parameters can be related to reliability or performance. Among the reliability-related parameters are mission reliability, availability and maintainability. Among the performance-related parameters are timing (maximum cycle time and minimum transport lag), throughput, and memory size. The requirements and objectives do not explicitly include signal processing or the human interface part of the control and display functions, although consideration is given to the interface requirements for these functions.

The requirements developed for use in the AIPS study are not intended to be an exhaustive statement of the requirements for any of the selected missions, but rather are intended to be representative of the broad range of requirements that are likely to be placed on systems that are candidates for the AIPS architecture.

A list of the requirements used in the current development is provided below. The requirements vary from mission to mission; the quantitative requirements indicated in parentheses below are obtainable by an AIPS implementation constructed using 1983 hardware and software technology [16]. The availability, cycle rate, I/O rate, memory, throughput, transport lag, and reliability capabilities of the AIPS are expected to improve along with the rapid improvements in reliability and performance of computing technology.

Adaptability

Availability (0.95 failure probability at 5 years with no repair)

Cost Effectiveness

Cycle Rate (up to 100 Hz)

Environment

Expandable I/O (up to 100 Mbit/sec)

Expandable Memory (up to 500 Mbyte)

Expandable Throughput (up to 100 MIPS)

Function Distribution

Graceful Degradation
Hardware N-fail-op
Low Life Cycle Cost
Low Transport Lag (as low as 5 ms)
Maintainability
Mission Reliability (10^{-9} at 10 hours with no repair)
Modularity
System Real Time Clock

Definitions of these requirements and the AIPS approach to fulfilling them are presented in Appendix A.

2.4 Attributes

Based on the AIPS design requirements and objectives a set of AIPS attributes was compiled. An attribute is an unambiguous statement of a characteristic of the AIPS. Attributes pertain to both the hardware and software building blocks for the AIPS architecture. More specifically, some of the more important attributes that the FTP possesses are as follows.

There is a common Fault Tolerant Processor architecture that can be easily tailored to meet the varying performance and reliability requirements for different criticality functions in a given application, or to meet different requirements for different applications. The required level of reliability can typically be achieved by choosing the redundancy level of the FTP from among duplex, triplex and quadruplex. The FTP architecture is designed such that changing the redundancy level does not impact the operating system, other system software, performance or the redundancy management approach.

A FTP has sufficient performance that several tightly coupled functions can be collocated in a single FTP, thus avoiding delays associated with inter-FTP communications. It is modular so that its interfaces to other FTPs and I/O devices can be easily adapted to the specific application environment. The real time operating system, redundancy management software and I/O user services are common to all FTPs so that they can be validated as part of the basic building block, the FTP. The FTP is also able to diagnose its internal faults autonomously and manage its redundancy locally.

An important attribute of the IC Network is that it is sufficiently reliable that critical information can be transmitted and received correctly between physically dispersed FTPs in the presence of arbitrarily malicious faults. It has enough performance to support real time sharing of information between non-collocated functions. It permits graceful growth to accommodate additional processing sites with minimal change in the basic hardware or software (system services) architecture. It is able to adapt to faults and reconfigure in real time.

The IC Network supports communication between FTPs of varying redundancy levels, with faults in lower redundancy level FTPs confined locally and not affecting communication between higher redundancy level FTPs. The contention for access to the communication mechanism is resolved using a fault tolerant distributed algorithm.

The qualitative attributes that are directly related to the mechanism that links a FTP to sensors, effectors and other I/O devices (I/O Network) are similar to those of the IC Network. The contention resolution issue is only applicable to shared I/O media and the quantitative reliability and performance requirements may be somewhat different from the IC Network requirements.

A list of the AIPS attributes as currently captured is presented below. A complete description of each attribute, the reason it is required, and the means of achieving it is presented in Appendix B.

- Bitwise Identical Inputs
- Bitwise Identical Outputs
- Byzantine Resilience
- Commercial Processors
- Common Mode Fault Tolerance
- Concurrent I/O, Computation
- Conventional User Interface
- Damage Tolerance
- Diagnosability
- FCR Faults Independent
- Flexible Function Allocation
- Function Migration
- Graded Redundancy
- I/O Net
- I/O Network Growth/Repair
- IC Network Growth/Repair
- Implementation Technology
- Independence of Faults
- InterComputer Net
- Low Component Failure Rate
- Low Fault Tolerance Overhead
- Low I/O Net Transport Delay
- Low IC Net Delay

Low System Services Overhead
On-line Memory
Portability of SW Tools
Prevalidated Building Blocks
Real Time Operation
Reconfigurability
Reconfigurable FTP
Repairability
RM Independent of Application
Shared Mass Memory
Simplex Programming Model
Software Fault Avoidance
Software Fault Tolerance
SW Development Environment
Testability
Timer-based Interrupt
Variable # Processing Sites
Variable Local I/O

2.5 Rules, Specifications, and Guidelines

After stating the attributes, the rules and guidelines for the AIPS building blocks were compiled. These rules and guidelines imply a set of AIPS attributes that will be reflected in any avionics system when correctly followed.

A rule is a principle which must be followed to assure that a higher-level AIPS attribute holds, a specification is an aggregate of rules which describe all relevant characteristics of an AIPS component, and a guideline is a statement of policy or philosophy based on Draper experience, along with a statement of the effects of deviating from the guideline.

An example of a rule is that there must be at least $3f+1$ Fault Containment Regions (FCRs) in a f -Byzantine Resilient FTP; another rule is that members of a FTP must reside in different FCRs. These rules when taken together result in a specification. Specifications must apply to physical objects which can be engineered, designed, constructed, and validated. In the AIPS, physical objects correspond to the various AIPS building blocks: the FTPs, the I/O Network, the IC Network, and the System Services Software.

Because of time limitations, the AIPS rules were aggregated into specifications for the FTP, the I/O Net, the IC Net, and the System Services Software. See Section 2.7.1 for

recommendations for improved aggregation schemes. The names of the rules associated with each system object are listed below. The complete specifications for the FTP, I/O Net, IC Net, and System Services Software are presented in Appendices C, D, E, and F, respectively.

FTP Rules:

2f+1 Inter-FCR Connectivity
2f+1 Redundant Copies
3f+1 FCRs
Bitwise Comparison of Outputs
Bitwise Identical Code
Copies in Separate FCRs
Dielectric Isolation
f+1 Round Input Distribution
FCR Synchrony
Hardware Data Exchange
Hardware Fault Tolerant Clock
Independent Clocking
Independent Power
Interchannel Links
Memory-Mapped I/O
One Net Interface/Channel
Physical Dispersion
Shared CP/IOP Hardware
Shared CP/IOP Memory
Task Watchdog Timer

I/O Net Rules:

Chained Transactions
Circuit-Switched Nodes
I/O Sequencer

IC Net Rules:

Circuit-Switched Nodes
IC Interface Sequencer
Reliable Contention Protocol
Reliable IC Communications

System Services Software Rules:

Error Traps

- Fast_FDIR
- FTP_FDIR
- I/O Net FDIR
- I/O Network Manager
- IC Comm Services
- IC Network Manager
- IC Source Congruency
- Layered System Services
- Partitioned Design
- Privileged Mode
- Synchronization Software
- System FDIR
- System Services on all FTPs
- Transient Fault Tolerance

An example of a guideline is that interactive consistency algorithms performed in hardware reduce a FTP's fault tolerant-specific overhead. A complete list of the guidelines for the building blocks is presented below. A description of each guideline, its rationale, and the means for achieving it are presented in Appendix G.

- Ada Language
- Copies in only $2f+1$ FCRs
- Extra Transactions
- Function Prioritization
- Hardware/Software Partition
- I/O Crosstrapping
- Latent Fault Detection
- Multiple Connections/IOP
- Multiple IOSs/FTP
- N-Version Software
- Non-BR Fault Tolerance
- Simultaneous Device Access
- Spare Link Cycling

2.6 Directed Graphs

Many mission requirements may be levied upon an avionics system. An architectural concept such as the AIPS must in turn have some approach to meeting each mission requirement. In some fashion any architectural approach therefore represents a

mapping from the mission requirements to a set of architectural rules and specifications which comprise the crucial tenets which must be followed to ensure that that architectural concept has been faithfully followed and the mission requirements have in fact been met.

The objective of this study is to formulate and demonstrate the use of a methodology which allows the expression, modification, and verification of this mapping, such that the relationships between the AIPS requirements, attributes, rules, specifications, and guidelines can be developed and displayed.

A Directed Acyclic Graph (DAG) was chosen to represent the vast amount of information and complex relationships between the AIPS requirements, attributes, rules, specifications, and guidelines. In the DAG representing the AIPS knowledgebase, each entity A_i , which may be a requirement, attribute, rule, or guideline, is viewed as a vertex in the DAG. Incoming edges emanate from entities which imply A_i and outgoing edges go to entities implied by A_i . The edges can represent many relationships. Some of these relationships are the following:

- "... A_i logically requires A_j ..."
- "... A_i is supported by A_j ..."
- "... A_i is achieved by A_j ..."

Each edge on the directed graphs is representative of one or more of the above relationships. All edges emanating from the entity which lead to attributes, rules and/or guidelines must be present in order to justify the entity.

The entities, A_i , are intended to constitute statements of fact or assertion about the architecture. The reason that a particular statement of fact must be made is identified at the source(s) of the edges which lead into that assertion; how that statement of fact is achieved is identified by the destination(s) of the edges emanating from the statement under consideration. Essentially, each assertion contains one or more pointers to the lower level assertions which logically support it and the higher-level assertions which justify it. For any given assertion, higher level assertions may be requirements, rules, guidelines, or attributes, while lower level assertions may be rules, guidelines, or attributes.

The use of this methodology is illustrated by an example; we will use the Graded Redundancy attribute. This example will serve to illustrate the methodology and introduce the reader to the format to be used for the knowledgebase in the Appendices.

The format begins with the title of the entity, in this case the Graded Redundancy attribute, followed by a discussion of the entity. At the current level of development of the knowledgebase, this discussion varies according to the amount of information that has been captured. It happens that the Graded Redundancy attribute has a relatively large amount of discussion attached; other entities have a somewhat more terse discussion. Future work is recommended to fill out the knowledgebase; this work is discussed in Section 2.7.1.

The question arises as to why an architecture should support Fault Tolerant Processors (FTP)s possessing varying redundancy levels. The rationale for the attribute is listed in the column entitled "Why Required," which contains the entries "Cost Effectiveness," "Mission Reliability," and "Modularity." Graded Redundancy supports the Cost Effectiveness requirement because it allows the designer of an avionics system to select the redundancy level appropriate to the mission's functionality, which may in turn vary for different functions of the mission. No more or less redundancy is forced upon the designer than is required to meet the mission's reliability goals, supporting a cost effective hardware implementation. Graded Redundancy also influences achieving a given Mission Reliability requirement because a given mission reliability is determined by, among other things, selection of the FTP redundancy level. Finally, Graded Redundancy supports the Modularity requirement because the modules, which are units of diagnosis and repair in the present context, are usually the redundant channels of a FTP; partitioning of the system into diagnosable repairable modules thus hinges on the capability for redundant operation.

Now that we have expressed what the concept of Graded Redundancy is and why it is needed in the AIPS architecture, we must express how the AIPS architecture achieves it. The means for achieving a concept is listed in the column "How Achieved," which in the Graded Redundancy example includes the entries "Identical FTP Design", "InterComputer Net," and "Simplex Programming Model." The first entity is a rule, "Identical FTP Design," a statement that the simplex, duplex, and triplex FTPs in the AIPS share an identical design; the only differences are in how many (one, two, or three, respectively) modules are "populated" with real hardware. A given module or channel need not be fully populated. The second attribute, "InterComputer Net," is an attribute which, among other things, states that lower redundancy level (lower-reliability) FTPs can be connected to higher redundancy level (higher-reliability) FTPs such that the lower-reliability FTPs cannot corrupt or impede communications between higher-reliability FTPs. The third attribute, "Simplex Programming Model," is a statement that the programmer's view of the AIPS FTP is that of a simplex, nonredundant processor regardless of its redundancy level.

The Graded Redundancy attribute, its rationale, and its means of achieving are depicted in graphical form in Figure 2-1.

Using this methodology, a top-level attribute graph was created (see Figure 2-2). This graph is an overview of the representation of relationships between the avionics requirements and the entire set of AIPS attributes. On the left of the graph, the avionics requirements listed in Section 2.3 are listed. Emanating from each requirement is a set of edges which terminate at the set of AIPS attributes, rules, or guidelines by which an AIPS meets that requirement. The top-level graph provides to the user an idea of the depth of knowledge that is involved when designing a system, and only includes the relationships between the avionics requirements and the first tier of attributes, rules, and guidelines. Additional relationships among the attributes, rules, and guidelines are depicted in lower-level directed graphs.

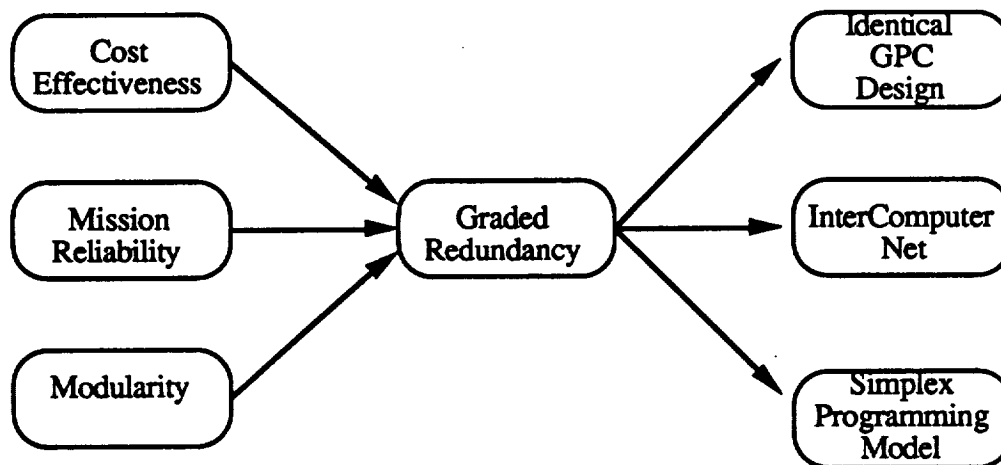


Figure 2-1. Graded Redundancy Attribute

From examination of the top-level graph it is clear that many complex relationships exist among the requirements and the lower-level attributes, rules, and guidelines of the AIPS. Moreover, the top-level relationship graph does not depict relationships among the lower-level entities themselves, nor how the lower-level entities are achieved by the AIPS architectural approach. These latter relationships are conceptualized in lower-level DAGs. On the lower-level directed acyclic graphs, the attributes which are mentioned on the top-level graph are described in detail.

This study does not include the lower-level DAGs in graphical format for a number of reasons. First, the current tools for manipulating and expressing the AIPS relationships require the manual construction of graphical depictions. To date, all this information has been manually produced and organized, a process requiring many man hours. In addition to being extremely tedious and error-prone, modification of the DAG is quite difficult and time-consuming. Second, the entire AIPS DAG would not fit legibly on a single page. Presentation of the entire DAG would therefore have required partitioning the DAG into numerous page-sized segments and constructing a complex inter-page cross-referencing scheme. Finally, it is thought that presentation of the raw graphs is an insufficiently rich explanatory format for the complex relationships existing among the graphical entities. These considerations and others influenced us to concentrate on the AIPS-specific job of capturing the AIPS knowledge and identifying the relationships, instead of identifying or constructing an appropriate knowledge representation and manipulation tool. However, to help alleviate the great amount of time and effort involved in future applications or extensions of this approach, an appropriate information management tool is required. Desirable capabilities of such a tool are discussed in Section 2.7.

Attribute: Graded Redundancy

In a system with multiple objectives (applications), the AIPS can be configured to achieve, in an economical way, the requisite probability of loss of each individual application function. In other words, only the appropriate level of redundancy required by each application has to be provided. The required level of reliability can typically be achieved by selecting the redundancy level of the FTP to be simplex, duplex, triplex, or quadruplex.

In the AIPS it is possible to vary the AIPS FTP redundancy level from simplex to quadruplex. The FTP hardware design is modular. The basic module is one channel which comprises two processor and their associated local memory, data exchange, shared bus controller, monitor/interlock, an I/O sequencer and an IC Interface Sequencer (Not applicable for centralized system). A duplex FTP consists of two such identical modules or channels, a triplex FTP has three modules and so on.

An ability to mix duplex, triplex, and quadruplex processors in the same system (and even simplex) without compromising the integrity of the higher redundancy processors or the communication between them, makes it possible to match the level of hardware redundancy to the required function reliability.

For duplex FTPs, it is necessary to invoke self tests in order to isolate the fault to one of the two channels. If a high degree of confidence is necessary for this fault isolation, it would be best not to do it on-line but to defer it for later maintenance. In a duplex system, on-line self tests cannot be relied upon to provide correct isolation of faults for every type of fault. That is, their isolation coverage is not 100 per cent. A far easier and higher coverage approach is to take the unit off line and swap cards to isolate a fault to the card level.

Why Required:

Cost Effectiveness
Mission Reliability
Modularity

How Achieved:

Identical FTP Design
InterComputer Net
Simplex Programming Model

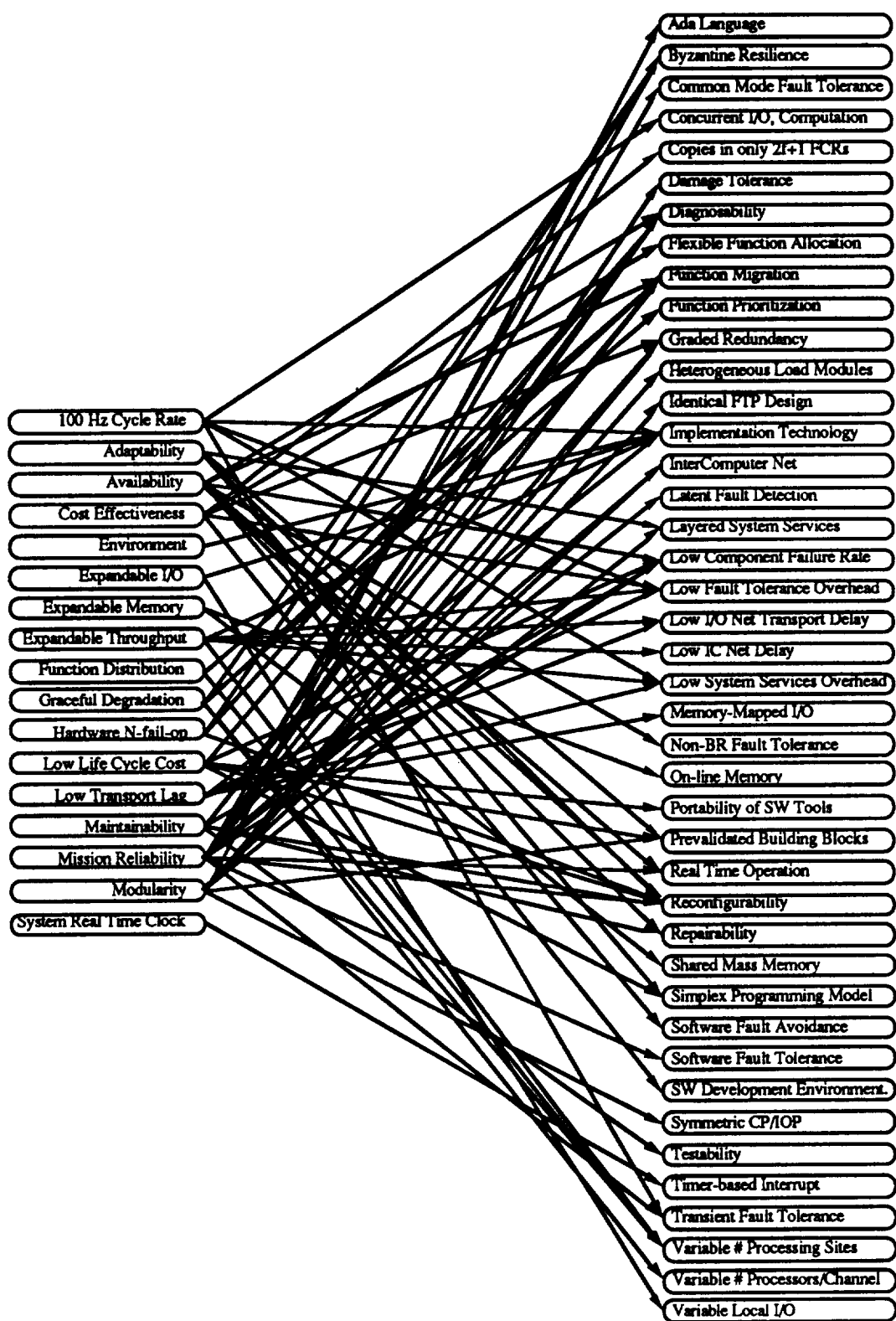


Figure 2-2. Top-Level Relationship Graph

2.7 Conclusions and Recommendations

2.7.1 Conclusions

This study has conceived and constructed a transitive relationship among the AIPS' mission requirements and architectural attributes, rules, and guidelines. The relationships among requirements, attributes, rules, specifications and guidelines are expressed through the use of a Directed Acyclic Graph (DAG) conceptualization.

It is thought that the elaboration of the set of transitive relationships among important architectural features has several benefits. The relationships depicted in the directed graphs allow a designer unacquainted with the details of fault tolerance technology to understand key issues regarding organization and operation of an AIPS. They also provide a framework which permits traceability of the AIPS mission requirements to the AIPS design specifications, a traceability which it is believed will enhance the AIPS' validatability. The approach also provides a pedagogy for the AIPS fault tolerance theory, and explains the rationale behind the AIPS approach to fault tolerance.

The experience of attempting to unambiguously state architectural characteristics and their relationships to mission requirements and each other has been extremely difficult yet illuminating. While far more remains to be done than has been done to date to generate a complete AIPS architectural specification and plausibly relate it to all mission requirements, it is felt that the DAG or a similar representation is useful in bringing order and causality to the process of mapping mission requirements to a computer system architecture.

2.7.2 Recommendations

Information from numerous AIPS publications and conversations with AIPS designers has been converted into the DAG format and is included in this document. However, it is felt that more information gathering and assimilation needs to be done than has yet been done.

For the knowledge currently captured, this work would comprise review of the assertions and their relationships and addition of more illuminating text regarding each one. An activity that has been relatively neglected in the current work is elaboration of the meaning of each edge in the DAG. This should be performed and would consist of formulating rigorous arguments that the set of assertions pointed to by the edge set emanating from a given attribute assures that the attribute holds, and a detailed explanation of the nature of the dependencies among assertions, including whether they are logical, quantitative, intuitive, etc.

Late in the present study confusion arose regarding the treatment of the AIPS building blocks. Entities such as the InterComputer Network were originally treated as

attributes, while later on it became obvious that they were "system objects" possessing attributes, rules, and guidelines of their own. Moreover, the mapping of system functions such as FDIR to the AIPS' system objects possesses attributes and must follow a set of rules and guidelines, and must be included into this methodology. The formulation presented herein therefore needs to be augmented to recognize that an AIPS instantiation comprises

system requirements, which correspond to Requirements in the current document,

system attributes, which correspond to Attributes in the current document,

system objects (FTP's, IC Networks, I/O Networks, ICIS's, IOS's, System Software Services), each of which has an aggregate of applicable rules and guidelines,

system functions (which may depend upon attributes possessed by several system objects: IC Net FDIR relies upon the system objects FTP's, IC Network, ICIS, and System Software Services),
and

mappings from functions to objects, which consists of a set of rules and guidelines.

It is our opinion that any continued development should be performed with this conceptual view as a starting point.

Additional knowledge needs to be converted into the DAG format. This document has developed and discussed one of the four major categories of information contained in the total AIPS Knowledgebase: the architecture guidelines and specifications and their relationship to mission requirements and AIPS attributes. It is recommended that the other AIPS knowledgebases be addressed. The information gathered on these knowledgebases needs to be organized and formally documented. All information in the knowledgebases then needs to be grouped together and presented as one transitively linked unit.

To a certain extent this could continue using the manual method of entering the knowledge and manually updating links between the assertions, as shown in the Appendices. Graphical depictions have been generated using the MacDraw tool, while a simple HyperCard application was written to facilitate assertion entry and management. However, our experience to date has indicated that a more automated and integrated information management tool would vastly facilitate the information manipulation required to use the DAG methodology. Based on our experience, the tool should have the following characteristics.

It should allow the entry and maintenance of an arbitrary amount of text regarding each assertion, and the entry and maintenance of the links between assertions. Furthermore, each link represents a relationship between two assertions; that relationship can be logical, qualitative, quantitative, rigorous, or intuitive. The nature of the relationship should be able to be expressed and updated. Edges emanate from a source assertion to other assertions, the aggregate of which are intended to guarantee that the source assertion holds. However, we have found that not all such edge sets form an "and" relationship; some destination assertions are merely recommendations (e.g., guidelines), while it is often the case that if one destination assertion is taken to an extreme (e.g., Component Failure Rate), then another destination assertion may be irrelevant (e.g., Reconfigurability). The information management tool must therefore be able to manage arbitrary "and/or" logical relations on the set of edges emanating from a source assertion. The logical relation may in fact change as a function of the quantitative values of the destination assertions, as in the example above. The tool should have a facility for consistency checking, that is, to ensure that there are no cycles in the graph.

The tool needs to have an explanation facility to allow a user to ask why an attribute is required, or how it is achieved. The explanation should be able to proceed to any level of transitivity desired by the user and included in the DAG. In particular, the tool should have the capability to propagate changes at the rule and guideline level back to the requirements level to allow the user to see the effect(s) of violating a rule or a guideline. In some cases, these effects will be quantitative, in others they will be failure to meet a requirement. The tool should have the capability to generate and display a graphical depiction of any subgraph of the DAG; the graphical depiction process has by far been the most tedious part of our current endeavor. It is our opinion that a tool capable of satisfying many of these needs should not be very difficult to construct or obtain.

The intended target of this methodology is avionics systems designers and users who are not necessarily experts in distributed fault tolerant systems. It is therefore important to provide to them the work we have done to date and allow them to comment on its organization, presentation, appropriateness of technical depth, and other issues important to them.

Appendix 2.A: Requirements

Representative requirements anticipated to be levied upon an AIPS are listed below. The AIPS capabilities listed below are based upon 1983 computing technology. The availability, cycle rate, I/O rate, memory, throughput, transport lag, and reliability capabilities of the AIPS are expected to improve along with the rapid improvements in reliability and performance of this technology.

Adaptability

Availability (0.95 failure probability at 5 years with no repair)

Cost Effectiveness

Cycle Rate (up to 100 Hz)

Environment

Expandable I/O (up to 100 Mbit/sec)

Expandable Memory (up to 500 Mbyte)

Expandable Throughput (up to 100 MIPS)

Function Distribution

Graceful Degradation

Hardware N-fail-op

Low Life Cycle Cost

Low Transport Lag (as low as 5 ms)

Maintainability

Mission Reliability (10^{-9} at 10 hours with no repair)

Modularity

System Real Time Clock

For each requirement listed above, we present a statement of that requirement and a list of attributes which the AIPS possesses which help to ensure that the requirement is met.

Requirement: Adaptability

The AIPS core architectural concepts possess minimal sensitivity to changes in functional requirements, performance goals, and levels of desired reliability and/or fault tolerance. It is possible to physically or functionally add to an implementation of the architecture while minimizing the effect on existing functions with which the added function has no communication.

How Achieved:

Layered System Services

Prevalidated Building Blocks

Reconfigurability
Simplex Programming Model

Requirement: Availability

The availability of a system at time t , $A(t)$, is the probability that the system is operational at time t given that it was operational at time 0. During the time interval $(0,t)$ repairs may have been performed. If the limit of this function exists as t goes to infinity it expresses the expected fraction of time that the system is available to perform useful computations.

Availability is typically used as a figure of merit in systems in which service can be delayed or denied for short periods without serious consequences.

The AIPS architecture shall provide for the achievement of high levels of operational availability for those applications such as commercial and military aircraft that require a high rate of mission executions to allow timely repair after a mission, to allow initiation of a mission with failed components, and to allow delay of maintenance until a more convenient time or place.

The AIPS can be designed to possess a 0.95 failure probability at 5 years with no repair.

Representative availability analysis results for the AIPS are presented in Section 4.

How Achieved:

Diagnosability
Function Migration
Low Component Failure Rate
Non-BR Fault Tolerance
Real Time Operation
Reconfigurability
Repairability
Software Fault Avoidance

Requirement: Cost Effectiveness

Cost effectiveness is an example of a qualitative requirement. The AIPS possesses several attributes which contribute to a cost effective implementation.

How Achieved:

Copies in only $2f+1$ FCRs
Flexible Function Allocation
Graded Redundancy
Low Fault Tolerance Overhead

SW Development Environment.

Requirement: Cycle Rate

The AIPS can support control loop cycle rates as fast as 100 Hz.

How Achieved:

Concurrent I/O, Computation
Implementation Technology
Low Fault Tolerance Overhead
Low System Services Overhead
Timer-based Interrupt

Requirement: Environment

The AIPS must be able to be implemented with components conforming to the appropriate environmental specification (i.e. radiation hardening, temperature, vibration/acceleration limits), and power, weight, and volume characteristics for each specific application without requiring revalidation of the core AIPS architectural concepts.

How Achieved:

Implementation Technology

Requirement: Expandable I/O

The AIPS architecture shall be capable of expansion of total external input/output to 100 Mbits/second (1983 computing technology).

How Achieved:

Implementation Technology
Variable Local I/O

Requirement: Expandable Memory

The AIPS architecture shall be capable of memory expansion to 500 million bytes (1983 computing technology). This memory can be divided into two general types: 1) on-line memory for the storage of currently executing or soon-to-be-executed programs, and 2) mass memory for the storage of programs and data for which immediate access is not required. The partition between these two types is application-specific.

How Achieved:

On-line Memory
Shared Mass Memory

Requirement: Expandable Throughput

The AIPS architecture shall be capable of throughput expansion to 100 MIPS (1983 computing technology). Total throughput refers to the combined processing capabilities which are available to support application functions.

How Achieved:

- Implementation Technology
- Low Fault Tolerance Overhead
- Low I/O Net Transport Delay
- Low IC Net Delay
- Low System Services Overhead
- Variable # Processing Sites

Requirement: Function Distribution

The distribution of application and system service functions among multiple FTPs shall be allowed to change, via function migration, in response to system requirements for resource/function reallocation or in response to component failures. The allowed allocation of function to processor resource shall include the static no-reallocation case.

How Achieved:

- Function Migration
- Variable # Processing Sites

Requirement: Graceful Degradation

Graceful degradation is the attribute whereby the system's functional capability degrades smoothly with the occurrence of faults. After a given and well-defined number of faults, the system is either still capable of performing all of its intended functions, or certain less-critical functions have been suspended and the most critical functions can still be performed.

How Achieved:

- Byzantine Resilience
- Function Migration
- Function Prioritization
- Variable # Processing Sites
- Variable # Processors/Channel

Requirement: Hardware N-fail-on

The hardware will continue to operate correctly after arbitrarily malicious faults in up to N separate FCRs.

How Achieved:

Byzantine Resilience
Diagnosability
Reconfigurability

Requirement: Low Life Cycle Cost

The AIPS possesses features which aid in reducing its life cycle costs.

How Achieved:

Diagnosability
Low Component Failure Rate
Mission Reliability
Portability of SW Tools
Prevalidated Building Blocks
Reconfigurability
Simplex Programming Model

Requirement: Low Transport Lag

The input transport lag of a system is defined to be the time elapsed from the availability of an input datum from a sensor to the delivery of that datum to an application task such as a control task. The output transport lag is defined to be the time elapsed from the assertion of an output from a computational task to the reception of the output by an output device such as an actuator or effector. The transport lag requirements for an AIPS system may range upwards from a lower bound 5 ms (1983 computing technology).

How Achieved:

Implementation Technology
Low Fault Tolerance Overhead
Low I/O Net Transport Delay
Low System Services Overhead
Memory-Mapped I/O

Requirement: Maintainability

Loosely speaking, maintainability may be defined as the "ease" with which a system may be kept in operational readiness over a given period of time.

Quantitatively, maintainability can be indicated by two parameters:

- MTBF: is the mean-time-between-failures for the system as a whole.
- MTTR: mean-time-to-repair, is defined to be the time from the moment of a component fault to the time it is repaired or replaced with a spare.

How Achieved:

Diagnosability
 Function Migration
 Low Component Failure Rate
 Reconfigurability
 Repairability
 Testability

Requirement: Mission Reliability

The Mission Reliability of a system at time t is defined to be the probability that the system can perform its intended function (in the presence of faults) over the time interval $(0,t)$. A system's Failure Probability is equal to one minus its Mission Reliability. It is typically assumed that repair of the system is not performed over the interval $(0,t)$.

Depending on the configuration of an AIPS instantiation, its failure probability can range from 10^{-4} per hour to 10^{-9} for a 10-hour mission without repair (1983 computing technology).

Representative reliability analysis results for the AIPS building blocks are presented in Section 4.

How Achieved:

Byzantine Resilience
 Common Mode Fault Tolerance
 Damage Tolerance
 Diagnosability
 Function Migration
 Graded Redundancy
 InterComputer Net
 Latent Fault Detection
 Low Component Failure Rate
 Real Time Operation
 Reconfigurability
 Software Fault Tolerance
 Transient Fault Tolerance

Requirement: Modularity

The AIPS hardware and software are decomposed into fine-granularity modules which are units of service, failure, diagnosis, repair and growth. Modularity is important to fault-tolerant systems because individual modules must be replaceable online. Keeping modules independent also makes it less likely that a failure of one module will affect the operation of another module. Having a way to increase performance by adding modules allows the capacity of critical systems to be expanded without requiring major outages to upgrade equipment.

How Achieved:

- Ada Language
- Prevalidated Building Blocks
- Graded Redundancy
- Heterogeneous Load Modules
- Identical FTP Design
- InterComputer Net
- Layered System Services
- Symmetric CP/IOP

Requirement: System Real Time Clock

The AIPS provides system-wide knowledge of real time with a clock accuracy and quantization commensurate with application-specific requirements. Accuracy is determined by the drift rate, update rate and non-deterministic IC communication. The local processor time is synchronized with the broadcast system time. Upon receipt of the periodic system time message from the global computer, the enclosed time value is compensated by known biases and the resultant value used to dedrift the local time. The quantization is determined by the FTP period.

How Achieved:

- Timer-based Interrupt
- InterComputer Net

Appendix 2.B: Attributes

List of Attributes:

- Bitwise Identical Inputs
- Bitwise Identical Outputs
- Byzantine Resilience
- Commercial Processors
- Common Mode Fault Tolerance
- Concurrent I/O, Computation

Conventional User Interface
Damage Tolerance
Diagnosability
FCR Faults Independent
Flexible Function Allocation
Function Migration
Graded Redundancy
I/O Net
I/O Network Growth/Repair
IC Network Growth/Repair
Implementation Technology
Independence of Faults
InterComputer Net
Low Component Failure Rate
Low Fault Tolerance Overhead
Low I/O Net Transport Delay
Low IC Net Delay
Low System Services Overhead
On-line Memory
Portability of SW Tools
Prevalidated Building Blocks
Real Time Operation
Reconfigurability
Reconfigurable FTP
Repairability
RM Independent of Application
Shared Mass Memory
Simplex Programming Model
Software Fault Avoidance
Software Fault Tolerance
SW Development Environment
Testability
Timer-based Interrupt
Variable # Processing Sites
Variable Local I/O

Attribute: Bitwise Identical Inputs

Via the source congruency function, each nonfaulty channel of a FTP is provided with bitwise identical inputs.

Why Required:

Bitwise Identical Outputs

How Achieved:

2f+1 Inter-FCR Connectivity

3f+1 FCRs

f+1 Round Input Distribution

FCR Synchrony

Attribute: Bitwise Identical Outputs

Nonfaulty copies of a redundant process have bitwise identical outputs.

Why Required:

Byzantine Resilience

How Achieved:

Bitwise Identical Code

Bitwise Identical Inputs

Attribute: Byzantine Resilience

AIPS FTPs which have a redundancy level of 2f+1 or greater can tolerate arbitrary (or Byzantine) failure behavior on the part of f Fault Containment Regions.

In general, up to "f" simultaneous arbitrary faults can be detected, masked, and identified with unity probability via bit-for-bit comparison of the outputs of copies of a redundant process.

An AIPS InterComputer (IC) Network of triplex or greater redundancy can tolerate arbitrary (or Byzantine) failure behavior on the part of any component connected to the network. This includes network nodes and subscribers to the network.

Why Required:

Graceful Degradation

Hardware N-fail-op

Implementation Technology

Mission Reliability

Real Time Operation

Reconfigurable FTP

Repairability

How Achieved:

2f+1 Redundant Copies

Bitwise Comparison of Outputs

Bitwise Identical Outputs

Independence of Faults

Attribute: Commercial Processors

The AIPS engineering model and other AIPS instantiations do not preclude the use of commercial or other commonly-used processors.

Current Processing Technology

CSDL has constructed a VLSI-based version of the FTP. It is a quadruplex FTP using the Motorola 68020 as the base processor. It contains two processors per channel with one processor devoted to computational functions and the other to I/O functions. This allows the VLSI FTP to operate as a stand alone computer in a hierarchical system architecture or as a part of a distributed system.

The engineering model of the basic VLSI FTP consists of three boards per channel: two identical processor cards and a shared hardware card. The processor cards contain the 68020 microprocessor, its Floating Point Co-Processor (68881), one Mbyte each of PROM and RAM memory, a processor specific Configurable Gate Array (CGA), and interfaces to two buses, a modified version of the VME bus which serves as the shared bus and a full specification VMX bus which serves as a private bus for such things as additional processor memory. The third card in the system, called the Shared Bus Controller, contains all of the shared hardware and an arbiter to control access to that hardware. The shared hardware includes the data exchange hardware, shared memory, the Fault-Tolerant Clock, a Real Time Clock and an interface to a full specification, industry standard VME bus which serves as the VLSI's AP bus. The data communicator is implemented with a 6000 gate equivalent 2.0 micron CMOS CGA.

Why Required:

SW Development Environment

How Achieved:

Implementation Technology

Attribute: Common Mode Fault Tolerance

The AIPS possesses limited resilience to common mode faults. Common mode faults can be caused by external influences such as Electro-Magnetic Interference (EMI), Electro-Magnetic Pulse (EMP), and cosmic radiation. These sources can affect redundant computations simultaneously and dilute, if not totally defeat, the protection provided by hardware replication. There is also the possibility of faults and errors being introduced in various phases of system design, specification, and implementation. They can manifest themselves as hardware, software or power-related failures in all of the redundant copies of the hardware.

Why Required:
Mission Reliability

How Achieved:
FTP_FDIR

Attribute: Concurrent I/O. Computation

The AIPS architecture is designed to allow decoupling of the computational and I/O streams of transactions. The CP is unburdened from performing any I/O operations. The application code runs on the CP while the I/O processing is done on the IOP. In addition the system is capable of running chains of transactions on an I/O Network and collecting data from sensors at the same time as the I/O Processor is performing its various functions.

Why Required:
Cycle Rate

How Achieved:
I/O Network Manager
Shared CP/IOP Hardware
Shared CP/IOP Memory
Symmetric CP/IOP

Attribute: Conventional User Interface

Complex programming and operations interfaces can be a major source of system failures. In the AIPS an attempt is made to simplify or automate interfaces to the system. An example is the use of ISO OSI-International Standards Organizations on Open System Interconnections. The OSI model of a communication system is composed of "layers." These layers contain the functions necessary to perform the communication, with certain restrictions on the functions in each layer. The layering groups function together to facilitate modifying any layer with minimal impact to the rest of the system.

Why Required:
Portability of SW Tools
SW Development Environment

How Achieved:
Layered System Services

Attribute: Damage Tolerance

The AIPS is able to tolerate limited damage events such as that caused by weapons or electrical shorts, overheating, or localized fire.

Why Required:

Mission Reliability

How Achieved:

3f+1 FCRs

I/O Net

Independence of Faults

InterComputer Net

Attribute: Diagnosability

The hardware components of an AIPS are fault tolerant building blocks such as FTPs, IC Network, and I/O Network. A fault occurring in any such building block is detected and diagnosed by the AIPS System Services. Faults are diagnosed to the following items:

FTP: Faults occurring in a FTP are diagnosed at least down to the channel containing the fault with 100% accuracy. In many cases it is possible to diagnose faults down to a particular module resident in that channel, with less certainty of accuracy. These modules may include the IOS, the ICIS, and other I/O devices.

I/O Net: Faults occurring in the I/O network are diagnosed down to either a faulty link or a faulty circuit-switched node. Because a single-layer I/O network is not Byzantine resilient faults are not diagnosed with 100% accuracy.

IC Net: Faults occurring in the IC Network are diagnosed down to either a faulty link or a faulty circuit-switched node. Because the IC Network is triple redundant and Byzantine resilient, faults can be diagnosed down to a layer with 100% accuracy. Diagnosis within a layer is not guaranteed to be 100% accurate.

Why Required:

Availability

Hardware N-fail-op

Low Life Cycle Cost

Maintainability

Mission Reliability

Repairability

How Achieved:

FTP_FDIR

I/O Network Manager

IC Network Manager

Spare Link Cycling

Attribute: FCR Faults Independent

If $P(A)$ is the probability that a fault exists in one FCR and $P(B)$ is the probability that a fault exists in any other FCR, then $P(AB)$, the probability that faults exist in both FCRs, is given by

$$P(AB) = P(A)P(B).$$

Why Required:

Independence of Faults

How Achieved:

Dielectric Isolation
Independent Clocking
Independent Power
Physical Dispersion

Attribute: Flexible Function Allocation

Many mappings of applications functions to the several processing sites (FTP's) of a distributed AIPS are possible. Selection of a desirable mapping is driven by input/output latencies, interfunction communication latencies, bandwidth, processing, and memory utilization considerations, and functions' accessibility to I/O and other needed resources. Flexible allocation of functions to the processing sites is achieved through the cross-straping of sensors, effectors, and other I/O devices across a number of computers through I/O Networks, through inter-FTP communication capability through the IC Network, and by the availability of all system services on all FTPs.

Why Required:

Cost Effectiveness

How Achieved:

InterComputer Net
I/O Crosstrapping
System Services on all FTPs

Attribute: Function Migration

The distributed implementation of the AIPS architecture comprises multiple Fault Tolerant Processors (FTP's). During routine operations the FTPs at various sites are assigned to perform a fixed set of functions, each computer doing a unique set of tasks. However, in response to some internal/external stimulus such as a fault or a change in mission mode, the FTPs can be reassigned to execute different sets of functions. This may be achieved by allowing some functions to migrate from one FTP site to another. Under

certain conditions, it may also result in some functions being suspended entirely for a brief time period or for the remainder of the mission. System Manager functions as well as application functions can be the target of function migration. For example, the IC Network manager, a part of the System Manager, could be relocated to an alternate processing site if the current host should suffer a failure of its interface into the IC Network. Similarly, the function migration process itself can also be relocated.

In most real-time control applications, executing a function requires not just an access to program memory and CPU but also inputs from various sensors and access to actuators, displays, etc. In order to migrate such a function, it is necessary to have access to these sensors, effectors, and displays from the alternate processing sites. This is accomplished by placing the required I/O devices on shared I/O Networks (regional or global). This customization of the AIPS architectural concept for a specific application must be done by systems engineers using reliability and performance tools. Other preplanning activity necessary at the time of the system design is the enumeration of all the events that trigger migration of a particular function and alternative locations of programs and data.

A layered approach to interfunction communication is used so that the actual resources used to implement a function are transparent to the applications programmer. System software modules (specifically, the IC Communication Services) know the current physical location of functions and provide the necessary interfunction communication.

A number of constraints must be observed to achieve function migration:

Alternate function partitions that are potential configurations resulting from function migration should be designed with consideration of performance margin requirements.

In order to migrate a function from one FTP to another, it is necessary to provide adequate cross strapping of I/O devices required by the function to multiple FTPs.

All function migrations are preplanned in AIPS.

A federated system may not reallocate any functions amongst processing sites in real time.

There shall only be one function migration task in process at any time.

Why Required:

Availability

Function Distribution

Graceful Degradation

Maintainability

Mission Reliability
Reconfigurability

How Achieved:

I/O Crosstrapping
IC Comm Services
System Services on all FTPs

Attribute: Graded Redundancy

In a system with multiple objectives (applications), the AIPS can be configured to achieve, in an economical way, the requisite probability of loss of each individual application function. In other words, only the appropriate level of redundancy required by each application has to be provided. The required level of reliability can typically be achieved by selecting the redundancy level of the FTP to be simplex, duplex, triplex, or quadruplex.

In the AIPS it is possible to vary the AIPS FTP redundancy level from simplex to quadruplex. The FTP hardware design is modular. The basic module is one channel which comprises two processor and their associated local memory, data exchange, shared bus controller, monitor/interlock, an I/O sequencer and an IC Interface Sequencer (Not applicable for centralized system). A duplex FTP consists of two such identical modules or channels, a triplex FTP has three modules and so on.

An ability to mix duplex, triplex, and quadruplex processors in the same system (and even simplex) without compromising the integrity of the higher redundancy processors or the communication between them, makes it possible to match the level of hardware redundancy to the required function reliability.

For duplex FTPs, it is necessary to invoke self tests in order to isolate the fault to one of the two channels. If a high degree of confidence is necessary for this fault isolation, it would be best not to do it on-line but to defer it for later maintenance. In a duplex system, on-line self tests cannot be relied upon to provide correct isolation of faults for every type of fault. That is, their isolation coverage is not 100 per cent. A far easier and higher coverage approach is to take the unit off line and swap cards to isolate a fault to the card level.

Why Required:

Cost Effectiveness
Mission Reliability
Modularity

How Achieved:

Identical FTP Design
InterComputer Net

Simplex Programming Model

Attribute: I/O Net

The AIPS provides a highly reliable communication path between FTPs and between a FTP and external I/O devices (sensors and effectors). AIPS provides a fault and damage tolerant network consisting of a number of full duplex links that are interconnected by circuit switched nodes to form a virtual multiplex bus. The network consists of a number of full duplex links that are interconnected by circuit switched nodes. In steady state, the circuit switched nodes route information along a fixed communication path, or virtual bus, within the network without the delays which are associated with packet switched networks. Once the virtual bus is set up within the network the protocols and operation of the network are similar to typical multiplex buses. Every transmission by any subscriber on a node is heard by all the subscribers on all the nodes just as if they were all linked together by a linear bus.

The circuit switched nodes are sufficiently intelligent to recognize reconfiguration commands from the network manager, which is resident in the FTP. The network manager performs the necessary diagnostics to identify the failed element and can change the bus topology by sending appropriate reconfiguration commands to the affected nodes.

The network performs exactly as a bus but is far more reliable and damage tolerant than a linear bus or ring. A single fault or limited damage can disable only a small fraction of the virtual bus, typically a node or a link connecting two nodes. Such an event does not disable the network, as would be the case for a linear bus. The network is able to tolerate such faults due to the richness of interconnections between nodes. By reconfiguring the network around the faulty element, a new virtual bus is constructed. The network can tolerate hardware faults, limited damage events including electrical shorts, overheating, localized fire and weapons effects. In addition, the network can be expanded very easily by adding more nodes linked to spare ports in existing nodes without shutting down the existing network. There are also no topological constraints on the network. In fact, these are simply subsets of the fault-tolerant network architecture. In effect, almost any bus configuration can be implemented using a circuit switched network. However, for proper operation, there can be no loops in the active network.

The AIPS I/O network is a reconfigurable, virtual bus which allows FTP subscribers to access input/output devices or Device Interface Units (DIUs) connected to the bus. The reconfigurability feature is allowed by the 5-ported nodes which join the various communications elements into a network. These nodes provide more than the minimum number of links required to form the bus. Under control of an I/O network manager, the spare links can be brought into service in response to a network failure, thus restoring service and increasing the reliability of the network.

The qualitative attributes that are directly related to the mechanism that links a FTP to sensors, effectors and other I/O devices (I/O Network) are essentially the same as for the IC Network. The contention resolution issue is only applicable to shared I/O media and the quantitative reliability and performance requirements may be somewhat different from the IC Network requirements.

Why Required:

Damage Tolerance

Reconfigurability

How Achieved:

Circuit-Switched Nodes

I/O Network Growth/Repair

I/O Network Manager

Spare Links

Attribute: I/O Network Growth/Repair

A Reconfiguration, Growth and Repair Algorithm for the I/O Network is necessary to be able to respond in real time to faults in the I/O Network.

A very robust network is established following the growth process. However, failures in the network hardware can occur at any time, and therefore it is necessary to periodically collect the status of the network nodes to verify their continued proper functioning. It is also necessary to be able to respond to communication errors detected when the network is used by I/O Communication Services to collect data for application programs. These tasks typically are scheduled to run at a much faster frequency than the network manager task. The ability to respond to these errors greatly reduces the response time for network repair.

The network manager is responsible for the maintenance of the bus in the presence of failures or damage to any of the nodes or connecting links. For the I/O Networks, errors are detected using techniques such as time outs, illegal protocol, and cyclic redundancy codes. For the IC Networks, errors are detected using techniques such as time outs, illegal protocol, cyclic redundancy codes and comparison and voting of the redundant layers. When an error is detected and identified, the failed element is removed from use and the bus is restructured using alternate devices or paths where available. Restore tests, which are performed on the network hardware, do not require the network to be taken out of service. The network manager can reconfigure the network to incorporate a failed part that has become operational. On line modifications can range from the reinstallation of a repair node to the addition of new nodes and links.

The I/O Network repair procedure varies according to the fault. If the fault is localized to a link, the network is repaired with spare link switching. If the fault is an active node failure (babbler), then regrowth of the network is required. If the fault is a node responding out of turn, the regrowth with testing is performed. In the case of regrowth and regrowth with testing, the repair of the network may take the network offline for more than one I/O cycle. Time critical applications may need a parallel network to meet their performance and reliability requirements.

Why Required:

I/O Net

How Achieved:

Circuit-Switched Nodes

I/O Network Manager

Attribute: IC Network Growth/Repair

The comments regarding the I/O Network Growth/Repair Attribute also apply to the InterComputer Network Growth and Repair attribute.

Why Required:

InterComputer Net

How Achieved:

Circuit-Switched Nodes

IC Network Manager

Attribute: Implementation Technology

Because the AIPS is an architectural concept (as opposed to an architectural implementation) which can be implemented in a variety of technologies, the AIPS can be constructed of advanced technology components such as high-speed processors, I/O networks, busses, and network nodes which aid in meeting performance and reliability requirements.

The AIPS architectural concept allows implementation in advanced hardware and software technologies without requiring revalidation of the core fault tolerance concepts.

Why Required:

Cycle Rate

Commercial Processors

Environment

Expandable I/O

Expandable Throughput

Low Component Failure Rate
Low Transport Lag
On-line Memory
Shared Mass Memory

How Achieved:

Byzantine Resilience
Simplex Programming Model

Attribute: Independence of Faults

Faults occurring in different copies of a redundant process are statistically independent.

Why Required:

Byzantine Resilience
Damage Tolerance

How Achieved:

Copies in Separate FCRs
FCR Faults Independent

Attribute: InterComputer Net

Physically dispersed FTPs communicate over a fault- and damage-tolerant redundant InterComputer Network. This network performs as a multiplex bus, but possesses reconfigurable circuit-switched nodes for fault and damage repair. In addition, multiple copies of this reconfigurable circuit-switched network are utilized to allow an additional degree of reconfiguration capability as well as fault masking communication paths among the separate FTPs. At least three copies of this network are required to achieve fault masking operation.

Some of the more important attributes for the IC Network are that it is sufficiently reliable that critical information can be transmitted and received correctly in the presence of arbitrarily malicious failures. It has enough performance to support real time sharing information between non-collocated functions. The communication mechanism is robust enough to support connections between physically dispersed FTPs. It allows for graceful growth to accommodate more processing sites with minimal change in the basic hardware or software (system services) architecture. It is able to adapt to failures and reconfigure in real time. The IC Networks support communication between FTPs of varying redundancy levels with failures of lower redundancy level FTPs confined locally and not affecting communication between higher redundancy level FTPs. The contention for access to the communication mechanism is resolved using a distributed algorithm.

Why Required:

Damage Tolerance
Flexible Function Allocation
Graded Redundancy
Mission Reliability
Modularity
Physical Dispersion
Reconfigurability
Shared Mass Memory
System Real Time Clock
Variable # Processing Sites

How Achieved:

Circuit-Switched Nodes
IC Comm Services
IC Network Growth/Repair
IC Network Manager
Reliable Contention Protocol
Reliable IC Communications
Spare Links

Attribute: Low Component Failure Rate

The AIPS architectural concept permits the selection and utilization of components having failure rates commensurate with the reliability, availability, maintainability, and cost requirements of a given mission. Variation of component failure rate does not affect the core architectural concepts of the AIPS; the effect of varying the component failure rates is instead reflected in the ultimate reliability of the system as indicated by the AIPS reliability models.

Why Required:

Availability
Low Life Cycle Cost
Maintainability
Mission Reliability

How Achieved:

Implementation Technology

Attribute: Low Fault Tolerance Overhead

The primary AIPS approach to fault tolerance is through the synchronous execution of processes which exhibit bitwise agreement in the absence of faults. The processes of synchronization, voting, and interactive consistency are mechanized in special-purpose

hardware, with the result that these functions consume under 5% of the throughput of a FTP.

In addition, the redundancy management functions of the AIPS System Services are designed to consume minimal temporal overhead as well, thus permitting the delivery of a majority of the system's throughput and bandwidth to the application, in turn allowing such constraints as the 5 ms transport lag to be met.

The AIPS employs a hardware implementation of the source congruency, fault detection and error masking function, and a mostly software implementation of fault isolation and reconfiguration functions. By performing fault detection and masking in hardware, the FTP minimizes the amount of time-critical software necessary to perform redundancy management. This specialized hardware is comparable to and in many cases less than the hardware complexity of the specialized interchannel communications hardware of more software-intensive solutions to fault tolerant processing. The voting and fault-handling hardware can be implemented in a single custom gate array and can be made compatible with any processor architecture or microprocessor chip.

The software implementation of fault isolation and reconfiguration functions, known as the FTP FDIR software, is split into several tasks. The fast task can be run frequently, while the more complex tasks are run only on demand or at a lower frequency. This division allows for high fault coverage while reducing the amount of processor time used. The Ada software task, Fast_FDIR, runs at a high frequency and is non-interruptible. This higher frequency of operation improves the reliability of the FTP by reducing the amount of time an error goes undetected. This combination of hardware- and software-implemented functionality results in a fault tolerance temporal overhead that is typically less than 10% of the FTP's throughput.

Why Required:

- Cycle Rate
- Cost Effectiveness
- Expandable Throughput
- Low Transport Lag
- On-line Memory
- Real Time Operation
- Resource Utilization
- Shared Mass Memory

How Achieved:

- Fast_FDIR
- Hardware Data Exchange
- Hardware Fault Tolerant Clock
- Hardware/Software Partition
- Shared CP/IOP Hardware

Attribute: Low I/O Net Transport Delay

A significant component of a system's transport lag is the time required to perform I/O network transactions. The AIPS attempts to reduce this lag via a number of architectural attributes.

Why Required:

Expandable Throughput
Low Transport Lag

How Achieved:

Chained Transactions
Circuit-Switched Nodes
Contention for Shared I/O
I/O Network Manager
I/O Sequencer
Shared CP/IOP Memory
Simultaneous Device Access

Attribute: Low IC Net Delay

Delays associated with inter-FTP communications may be minimized by collocating tightly-coupled functions on a single FTP. There is no inherent limitation on the number of functions that can be supported in a single FTP other than that resulting from memory, throughput, I/O, or application-specific imposed restrictions.

When, due to throughput, memory, or bandwidth overconsumption, a function must spill over into multiple FTPs, the AIPS attempts to reduce the inter-function latency via a number of attributes.

Why Required:

Expandable Throughput

How Achieved:

IC Interface Sequencer
IC Comm Services

Attribute: Low System Services Overhead

The AIPS system software provides numerous services in support of user applications. These services include functions invoked explicitly by applications programs, as well as functions performed autonomously by the system to maintain proper operation.

The AIPS operating system consists of the vendor supplied Ada Run Time System (RTS) along with those extensions needed to implement the functions of task execution management, memory management, intertask communications and software exception handling. The extensions are written in Ada with time critical sections done in assembly language to reduce system overhead. The AIPS operating system resides on every IOP and CP.

In the AIPS, I/O devices are accessed through I/O System Services whose purpose is to present a uniform interface between the application and the I/O devices. This is intended to buffer the programmer from the complexity of operating the I/O device, performing fault detection and redundancy management, and other functions. The AIPS implementation of these functions is designed to consume minimal temporal overhead.

The following data are representative of performance figures achieved using the CSDL-modified Ada RunTime System (RTS). The 68010 figures are for the AIPS Engineering Model FTP (68010, 7.8 MHz, Verdex Ada 5.4), and the 68020 figures are for the CSDL VLSI FTP (68020, 14.5 MHz, Verdex Ada 5.4).

Modified Ada Delay Scheduling: 1300 microseconds

Modified Delay Dispatch: 1200 microseconds

Simple Rendezvous: 500 microseconds (68010), 125 microseconds (68020)

Timer Dispatch: 2100 microseconds (68010), 540 microseconds (68020)

Context Switch: 480 microseconds (68010)

Local Event Dispatch: 1000 microseconds (68010), 245 microseconds (68020)

Remote Task (Event) Dispatch: 1460 microseconds (68010), 465 microseconds (68020)

Shared Memory Protected Access: 650 microseconds (Access - one CPU), 300 microseconds (Rupt handler - other CPU)

AIPS FTP Operating System Overhead is 23% of a 40 millisecond frame.

CSDL VLSI FTP Operating System Overhead is 6% of a 40 millisecond frame.

Why Required:

Cycle Rate

Expandable Throughput

Low Transport Lag

On-line Memory

Real Time Operation

How Achieved:

Implementation Technology
IC Comm Services
I/O Network Manager

Attribute: On-line Memory

The amount of on-line memory available to an AIPS application suite can be varied by varying the number of processing sites in the AIPS or by varying the amount of memory per channel of an AIPS FTP. Because of reasonably low memory consumption by the AIPS System Services and fault tolerance-related functionality, most of this memory can be made available to the application. The amount of memory available within a given weight, power, and volume envelope varies as technology progresses. Because the AIPS architectural concept is invariant with respect to implementation technology, these advances in technology can be incorporated into advanced AIPS configurations.

Why Required:

Expandable Memory

How Achieved:

Implementation Technology
Low Fault Tolerance Overhead
Low System Services Overhead
Variable # Processing Sites
Variable Memory/Channel

Attribute: Portability of SW Tools

Existing software development, debugging, and validation tools can be utilized to program an AIPS.

Why Required:

Low Life Cycle Cost
SW Development Environment

How Achieved:

Conventional User Interface

Attribute: Prevalidated Building Blocks

AIPS is a multicomputer architecture composed of hardware and software building blocks. These are fault tolerant processors, fault and damage tolerant inter-computer and input/output networks, a fault tolerant mass memory, a fault tolerant power distribution system and system software. Each AIPS building block is designed so that it can be vali-

dated individually. These building blocks are tested and validated before being incorporated into a system.

Why Required:

Adaptability
Low Life Cycle Cost
Modularity

How Achieved:

Fault Tolerant Processor
I/O Network
InterComputer Network
System Software Services

Attribute: Real Time Operation

One component of a computational system's reliability and/or availability requirement is that it meet real-time deadlines with a high probability. Failure to meet such deadlines can represent system loss, also known as dynamic failure.

The AIPS system is capable of real-time operation, that is, of predictable and deterministic response to events. These events may be timer interrupts or external environmentally-induced events such as an I/O event, fault occurrence, or unanticipated mission mode changes. Real-time behavior is guaranteed in the presence of arbitrary behavior on the part of a subset of the AIPS components.

Why Required:

Availability
Mission Reliability

How Achieved:

Byzantine Resilience
Low Fault Tolerance Overhead
Low System Services Overheads
Timer-based Interrupt

Attribute: Reconfigurability

Upon occurrence of component faults the AIPS may reconfigure its components to replace the failed component or to isolate it from the rest of the system. The component may be replaced by backup spares. Alternatively, it may simply be switched off and the system capability to execute all of its application functions degraded.

System objects which may be reconfigured are:

- Fault Tolerant Processors
- InterComputer Network
- I/O Network
- Application Functions (function migration)

Why Required:

Adaptability

Availability

Hardware N-fail-op

Low Life Cycle Cost

Maintainability

Mission Reliability

How Achieved:

Function Migration

I/O Net

InterComputer Net

Reconfigurable FTP

Attribute: Reconfigurable FTP

A redundant Fault Tolerant Processor (FTP) is capable of surviving, detecting, identifying, and reconfiguring from a fault which causes any single component to exhibit arbitrary (i.e., Byzantine) behavior, with a probability approaching unity.

The reconfiguration (or downmoding) process consists of setting vote and clock masks in the interchannel data exchange hardware and hardware fault tolerant clock such that the outputs of the faulty channel are not considered in votes and synchronization. The monitor interlocks are set such that the outputs of the channel of the FTP determined to be faulty is disabled.

- A quadruplex redundant FTP is reconfigurable to be fail-op-to-triplex/fail-op-to-duplex/fail-safe-to-simplex;
- A triplex redundant FTP is reconfigurable to be fail-op-to-duplex/fail-safe-to-simplex;
- A duplex redundant FTP is reconfigurable to be fail-safe-to-simplex;
- A simplex FTP cannot be reconfigurable to a safe state with any degree of certainty.

Why Required:

Reconfigurability

How Achieved:

Byzantine Resilience

Hardware Data Exchange
Hardware Fault Tolerant Clock

Attribute: Repairability

In the AIPS, a component diagnosed as failed can be replaced, thus effecting repair.

As with detection, repair can be either on-line or off-line. In off-line repair, either the failed component is not necessary for system operation, or the entire system must be brought down to perform the diagnosis and repair. In on-line repair, the component may be replaced immediately by a backup spare in a procedure equivalent to reconfiguration, or operation may continue without the component, as is the case with masking redundancy or graceful degradation. In either case of on-line repair, the failed component may be physically replaced or repaired without interrupting system operation.

The AIPS approach to achieving repairability is indicated by the attributes listed below. In addition, those attributes which support the Modularity Requirement also assist in achieving the Repairability Attribute.

Why Required:

Availability
Maintainability

How Achieved:

Byzantine Resilience
Diagnosability

Attribute: RM Independent of Application

In the AIPS, the Redundancy Management (RM) software is an integral part of the System Software Services and is not visible or accessible to the applications programmer. As part of the System Software Services building block, the RM software has been prevalidated by comprehensive examination and testing. Its correctness of execution is therefore independent of that of the application software. Similarly, validation of the correctness of the application software is independent of the AIPS RM strategy or software.

Why Required:

Simplex Programming Model

How Achieved:

FTP_FDIR

Attribute: Shared Mass Memory

In addition to the memory contained in each FTP, certain AIPS applications will require an amount of memory to be shared among several FTPs. This is referred to as the

Shared Mass Memory Attribute. This memory resides on the InterComputer Network in a manner similar to a FTP.

Why Required:

Expandable Memory

How Achieved:

Implementation Technology

InterComputer Network

Low Fault Tolerance Overhead

Attribute: Simplex Programming Model

A change in the redundancy level of an AIPS FTP does not impact the application software. Regardless of a FTP's redundancy level, the programmer's view of that FTP is that of a highly reliable, nonredundant, "simplex" FTP.

Why Required:

Adaptability

Graded Redundancy

Implementation Technology

Low Life Cycle Cost

SW Development Environment

How Achieved:

RM Independent of Application

Attribute: Software Fault Avoidance

The AIPS approach to software fault avoidance primarily relies upon a well-tested, familiar software development environment. The use of such an environment is possible because the AIPS approach to hardware fault tolerance is independent of the application software development process.

Why Required:

Availability

Mission Reliability

How Achieved:

Ada Language

SW Development Environment

Attribute: Software Fault Tolerance

The AIPS possesses architectural means to assist in tolerating faults in the System Services software and the applications software.

Why Required:

Mission Reliability

How Achieved:

Ada Language

Error Traps

FTP FDIR

N-Version Software

Non-BR Fault Tolerance

Partitioned Design

Privileged Mode

Task Watchdog Timer

Attribute: SW Development Environment

AIPS application software can be developed and validated using tools, compilers, and development stations already available for simplex, nonredundant processors. This is because the fault tolerance of the AIPS FTP is transparent to the application programmer, and because the AIPS architectural concept supports the use of commercially available processors. Because the software development process exactly duplicates the process which occurs in a simplex machine, it is necessary to have only a debug capability for a simplex channel. An AIPS is programmed in the Ada language using a validated Ada compiler, which provides standards for software interfaces.

Why Required:

Cost Effectiveness

Software Fault Avoidance

How Achieved:

Ada Language

Commercial Processors

Portability of SW Tools

Simplex Programming Model

Attribute: Testability

The AIPS architecture is designed to be testable in the sense that the operability of each major system component (CP/IOP, Data Exchange hardware, networks, and software) can be ascertained with a high degree of certainty.

Why Required:

Maintainability

How Achieved:

CP/IOP Testability

Data Exchange Testability

Error Logging

Network Testability

Performance Logging

Software Testability

Attribute: Variable # Processing Sites

The distributed version of the AIPS comprises two or more FTPs, interconnected by the InterComputer (IC) Network. The number of FTPs in a distributed AIPS is determined by the application's throughput, memory, reliability, and physical dispersion requirements. The number of FTPs in a distributed AIPS is upper-bounded by contention effects on the IC Network.

Why Required:

Expandable Throughput

Function Distribution

Graceful Degradation

On-line Memory

How Achieved:

InterComputer Net

Attribute: Variable Local I/O

The I/O network can be expanded by adding more nodes linked to spare ports in existing nodes. Nodes and subscribers to the new nodes (I/O devices or FTPs) can even be added without shutting down the existing network.

Why Required:

Expandable I/O

How Achieved:

Extra Transactions

Multiple Connections/IOP

Multiple IOSs/FTP

Appendix 2.C: FTP Specification

FTP Rules:

- 2f+1 Inter-FCR Connectivity
- 2f+1 Redundant Copies
- 3f+1 FCRs
- Bitwise Comparison of Outputs
- Bitwise Identical Code
- Copies in Separate FCRs
- Dielectric Isolation
- f+1 Round Input Distribution
- FCR Synchrony
- Hardware Data Exchange
- Hardware Fault Tolerant Clock
- Identical FTP Design
- Independent Clocking
- Independent Power
- Interchannel Links
- Memory-Mapped I/O
- One Net Interface/Channel
- Physical Dispersion
- Shared CP/IOP Hardware
- Shared CP/IOP Memory
- Symmetric CP/IOP
- Task Watchdog Timer
- Timer-based Interrupt

Rule: 2f+1 Inter-FCR Connectivity

Each FCR must be connected to each other FCR through at least 2f+1 disjoint communication paths.

All inter-FCR connections are made using line drivers, receivers and transceivers together with appropriate isolation resistors and point-to-point transmission lines with associated line termination networks. This electrical isolation prevents a fault in any given FCR from migrating past the boundaries of that FCR and corrupting other FCRs.

Why Required:

Bitwise Identical Inputs

Rule: 2f+1 Redundant Copies

At least 2f+1 copies of a redundant process are required for fault masking.

Why Required:
Byzantine Resilience

Rule: $3f+1$ FCRs

To tolerate f simultaneous Byzantine faults, a FTP must possess at least $3f+1$ Fault Containment Regions.

Processors, memories, and buses are replicated to achieve a very high degree of reliability and fault tolerance. The redundant elements are operated in tight synchronism which results in exact replication of computations and data. Fault detection coverage with this approach is one hundred per cent once a fault is manifested.

The hardware redundancy is built into AIPS to provide protection against random hardware faults. The number of such faults that can be tolerated depends on the level of redundancy. A triplex FTP can detect and identify a single fault with 100 per cent coverage. If two such faults occur nearly simultaneously in two processor channels, whether or not they are identical faults, the coverage drops much below a 100 per cent. This near simultaneity can be defined more precisely in terms of a time window which is dependent on the region affected by the fault. Within a triplex FTP, this window of vulnerability is the order of tens of milliseconds. A certain amount of time is required for the software to react to the manifestation of a hardware fault and identify and recover from it before the next fault manifests itself. This reaction time forms the window during which the system is vulnerable to a second fault.

Why Required:
Bitwise Identical Inputs
Damage Tolerance

Associated Guideline:
Copies in only $2f+1$ FCRs

Rule: Bitwise Comparison of Outputs

Each copy of a redundant process must compare its output with that of the other copies to achieve fault detection, masking, and identification.

Why Required:
Byzantine Resilience

How Achieved:
Hardware Data Exchange

Rule: Bitwise Identical Code

Each channel of a FTP possesses one or more processors. To achieve bitwise identical outputs in the absence of faults, processor replicates (CP, IOP, etc.) residing in different channels must execute bitwise identical instructions.

Why Required:

Bitwise Identical Outputs

Rule: Copies in Separate FCRs

Copies of redundant executions must reside in separate Fault Containment Regions. For example, if it is desired to realize a redundant, fault tolerant Computational Processor (CP), then copies of the CP must reside in separate FCRs.

Why Required:

Independence of Faults

Rule: Dielectric Isolation

All inter-fault set connections are made using line drivers, receivers and transceivers together with appropriate isolation resistors and point-to-point transmission lines with associated line termination networks. This electrical isolation prevents a fault in any given FCR from propagating past the boundaries of that FCR and causing fault to occur in other FCRs.

Why Required:

FCR Faults Independent

Rule: $f+1$ Round Input Distribution

Simplex source congruency is defined as congruent or identical distribution of data from a simplex source to a redundant system. This simplex source of data may be within or without the system. It is important that all redundant copies of hardware receive congruent values of data originating in the simplex source. This is a necessary condition in fault tolerant systems that rely on exact replication and comparison of computational streams.

One must perform source congruency operations on all simplex data coming into a redundant computer to prevent single point failures in the design. It is not sufficient to distribute simplex data to redundant elements in one step. To tolerate a single Byzantine fault, the redundant elements must exchange their copy of the data with each other to make sure that every element has a congruent value of the simplex data. This rule generalizes to a requirement that, to tolerate f simultaneous Byzantine faults, the redundant elements must perform $f+1$ interchannel exchanges of data.

Why Required:

Bitwise Identical Inputs

How Achieved:

Hardware Data Exchange

Rule: FCR Synchrony

Process replicates residing in different Fault Containment Regions must be synchronized to within a known temporal skew.

Why Required:

Bitwise Identical Inputs

How Achieved:

Hardware Fault Tolerant Clock

Synchronization Software

Rule: Hardware Data Exchange

The AIPS FTP possesses a data exchange (DX) mechanism that complies with the Byzantine Generals Problem requirements for number of FCRs, connectivity, rounds of communication and bounded skew.

The data exchange hardware in the AIPS FTP provides some of the most important functions from the viewpoint of fault tolerance and redundancy management. This hardware, in conjunction with the Fault Detection, Isolation, and Reconfiguration (FDIR) software detects hardware faults in the FTP, isolates them to at least a single channel and in many cases to one card in a channel, and takes appropriate action to prevent errors from propagating to other channels within the FTP as well as to devices and computers outside the FTP. The data exchange hardware, in conjunction with the I/O system services software, provides a congruent flow of data from sensors, displays, actuator feedbacks, and other devices on the I/O network to the redundant channels of the FTP. It also facilitates transmission of voted results to devices on the I/O network.

A variable number of redundant copies of transmitted data, depending on the source computer redundancy level, must be received by each channel of the FTP, processed for errors, and then in some cases voted across the FTP channels before they can be passed onto the user of that data. The data exchange hardware, in conjunction with the IC communication services, performs these functions and provides a congruent flow of data from other FTPs on the IC Network to the redundant channels of the FTP. It also facilitates transmission of voted results to FTPs on the IC Network.

The cross-channel communication and majority voter/source selection logic for the triplex FTP is used. This unique hardware is designed to support the theoretical requirements for exact consensus in the presence of arbitrary faults. The communicator data exchange hardware in each channel resides within a fault containment region. An interstage in each channel, with its independent timing and voltage references, forms another fault containment region. Thus, there are at least six fault containment regions in a triplex FTP.

The data exchange hardware is used to exchange data, the FTC signals, and external interrupts between channels. Data is exchanged among redundant channels on point-to-point links. The data exchange hardware also performs the bit-for-bit voting, fault detection and masking functions in a manner that satisfies all the requirements to protect it from Byzantine failures. The data exchange voter compares the three input streams on a bit-by-bit basis and produces a "majority" output bit for each input bit. The majority output of the voting process is compared to each of the three inputs and if any disagreements are detected, the identity of the disagreeing channel is stored in an error register, called the Error Latch. There are three bits in this latch: one bit for each of the three channels. In the presence of a single fault, the voter produces the correct result and latches the identity of the failed channel. As the name implies, this error information stays latched until cleared explicitly. The error latches are, in fact, read as memory mapped registers by the FDIR software, a critical part of the FTP operating system. A read operation automatically clears the error latches so that every time the FDIR software reads them, it obtains new error information. Between reads all error indications are OR'ed.

The data exchange hardware appears as a set of four transmit registers and one receive register on the IOP-CP shared bus. One transmit register is for voted exchanges and the other three are for simplex exchanges. For simplex source exchanges, data from the selected channel is transmitted to the three interstages. The interstages rebroadcast this data to every processor. The three copies received by each processor are voted in hardware on a bit-by-bit basis and deposited in the receive register. For voted exchanges, each channel writes to its own interstage. The second half of the operation is the same as for a simplex exchange. In both cases, the exchange hardware masks any single faults while voting on three copies of the data and also records the source of any faults in error latches. The basic architectural philosophy is to implement the fault detection and error masking functions in hardware since these functions are performed very frequently.

Triplex FTPs can isolate faults to one of three channels with high coverage using the data exchange hardware.

It is necessary to uncover latent faults in the voter and error latch hardware by periodically writing erroneous data patterns and verifying that the erroneous pattern is detected by the hardware.

Why Required:

Bitwise Comparison of Outputs
f+1 Round Input Distribution
Hardware/Software Partition
Low Fault Tolerance Overhead
Reconfigurable FTP

Rule: Hardware Fault Tolerant Clock

Identical processes executing on redundant hardware channels eventually diverge due to minor differences in the hardware timing of redundant channels. The AIPS approach to synchronizing redundant hardware elements is to phase lock their clocks and to design the hardware to be clock deterministic. Such an approach provides continuous synchronism in the background without any intervention on the part of the software.

The FCRs of a FTP are kept in synchrony by a distributed high-speed fault tolerant clock signal which directly feeds the FCR's clock network. Hardware implementation of the synchronization function relieves the operating system or applications software from the overhead and complexity of performing algorithms to synchronize the redundant channels.

Why Required:

FCR Synchrony
Independent Clocking
Low Fault Tolerance Overhead
Reconfigurable FTP

Rule: Identical FTP Design

The simplex, duplex, and triplex FTPs in the AIPS share an identical design. The only differences are in which modules are "populated" with real hardware. A given module or channel need not be fully populated.

Why Required:

Graded Redundancy
Modularity

Rule: Independent Clocking

Each nonfaulty Fault Containment Region must possess a digital clock signal which is guaranteed to be within a known, given skew of every other nonfaulty Fault Containment Region, even in the presence of Byzantine faults in the fault tolerant clocking circuitry.

Why Required:

FCR Faults Independent

How Achieved:

Hardware Fault Tolerant Clock

Rule: Independent Power

The FCRs comprising a FTP are powered by the Diode-OR output on their own channel's power and that of an adjacent channel, such that the failure of a single power supply cannot result in a single point failure for the FTP.

Why Required:

FCR Faults Independent

Rule: Interchannel Links

The channels of a redundant FTP are connected by dedicated point-to-point communication links. Interchannel links can be several meters long, and the channel interface hardware is designed such that long links do not pose a problem in synchronizing and communicating among dispersed channels.

Why Required:

Physical Dispersion

Rule: Memory-Mapped I/O

Memory-mapped I/O is the conventional type of I/O configuration wherein external devices interface directly with FTP buses and are addressed directly by the FTP. A CP or IOP may have local dedicated I/O devices that can be accessed directly by the CP or IOP as memory locations. The memory mapped I/O may consist of local switches, discretes, A/Ds, D/As and interrupt driven devices. It is likely that I/O devices possessing a 5 ms transport lag requirement will reside as memory-mapped devices.

To meet extremely low transport lag requirements, it is necessary to provide dedicated I/O devices to the CP itself. That is, the CP would directly read the sensors without going through an IOP or a local processor, perform the required control law computation, and write directly to actuators.

Why Required:

Low Transport Lag

Rule: One Net Interface/Channel

A network has at most one interface per FTP channel, i.e. redundant root links to a network from a FTP come from distinct channels. Thus the maximum number of network interfaces connecting a FTP to a network is equal to the number of channels in the FTP.

Why Required:

Reliable Contention Protocol
Simultaneous Device Access

Rule: Physical Dispersion

Channels of a FTP must be physically dispersed. Multiple FTPs may also be physically dispersed, in which case they shall communicate over the redundant InterComputer network.

Why Required:

FCR Faults Independent

How Achieved:

Interchannel Links
InterComputer Net

Rule: Shared CP/IOP Hardware

To reduce the overheads of fault tolerance, the data exchange hardware as well as hardware that provides access to I/O and IC networks and a real time clock, are shared between the CP and the IOP. This reduces the fault tolerance overhead by half as compared to having separate exchange mechanisms for both the CP and the IOP.

Why Required:

Concurrent I/O, Computation
Low Fault Tolerance Overhead

Rule: Shared CP/IOP Memory

The IOP and CP exchange data through a shared memory. The IOP and CP have independent operating systems that cooperate with each other. The IOP and CP actions are therefore synchronized to some extent. To help achieve this synchronization in software, a hardware feature enabling one processor to interrupt the other processor has been provided. One way to implement the interrupt mechanism in hardware is to assert an interrupt when some block of dual-port memory is written into. This would allow the processors the flexibility of causing interrupts for a variety of reasons, the actual reason and message being transmitted through the contents of the shared memory.

Why Required:

Concurrent I/O, Computation
Low I/O Net Transport Delay

Rule: Symmetric CP/IOP

The AIPS FTP architecture is symmetric from the viewpoint of the processors. Since both the CP and IOP have access to all the external interfaces, the FTP can be operated with both or only one processor per channel. Therefore, in AIPS applications that do not have intensive I/O and/or IC communication, one processor per channel may suffice. The shared bus controller card, along with the dedicated data exchange registers, can be removed leaving only a single processor FTP with only the core data exchange hardware.

In the AIPS it is possible to vary the number of IOS interfaces per FTP and per channel to fit various processor and network redundancy levels and parallel and partitioned networks. A given module or channel need not be fully populated. For example, a triplex FTP may have only two I/O sequencers rather than three.

Why Required:

Concurrent I/O, Computation
Modularity

Rule: Task Watchdog Timer

The watchdog timer is provided to increase fault coverage and to cause the processor to fail-safe in case of hardware or software malfunctions. The watchdog timer resets the processor and disables all of its outputs if the timer is not reset periodically.

Why Required:

Software Fault Tolerance
Software Testability

Rule: Timer-based Interrupt

Each channel of a FTP possesses a hardware timer which is capable of interrupting the channel upon the expiration of a programmable time interval. Because of tight synchrony of the channels and clock deterministic behavior on the part of the FTP channels, these interrupts will occur at very close to the same time in all channels and will interrupt each channel at identical points in their replicated instruction streams.

Why Required:

Cycle Rate
Real Time Operation
System Real Time Clock

Appendix 2.D: I/O Net Specification

I/O Net Rules:

Chained Transactions
Circuit-Switched Nodes
I/O Sequencer

Rule: Chained Transactions

I/O requests are specific to one I/O service. They consist of a set of chains, at most one per network within the service. All chains in an I/O request are started at the same time. The I/O request is completed, and data becomes available to a user, when all chains within the request are completed.

Chains allow efficient use of the communications bandwidth, but are only applicable to a single network. Accordingly, the user must specify the transactions that will form the chain, and the network on which they will be executed.

There are two chains of transactions used by the I/O network managers: a monitor chain and a command chain. The monitor chain, used to monitor the status of all the nodes in the network, must contain at least one transaction for each node in the network. The command chain, used to send commands to any two nodes in the network, must contain at least two transactions for each node in the network. Within the command chain, only two transactions are selected for execution at any one time based on the nodes to which the commands are addressed and the order in which they are to be commanded. For the monitor chain, each transaction corresponding to a node in the network is executed.

Why Required:

Low I/O Net Transport Delay

Rule: Circuit-Switched Nodes

The AIPS contains 5-ported nodes which join the various communications elements into a network. These nodes provide more than the minimum number of links required to form a single communication bus. Under control of a network manager (either I/O or IC), ports on any given node can be activated or deactivated, allowing spare links can be brought into service in response to a network failure or for testing.

In steady state, the communication path operates as if it were a conventional, time division multiplex bus. It differs from a linear bus in that the data is routed by circuit switched nodes along one of several possible paths. Each node in a properly configured, fault free network receives transmissions on exactly one of its enabled ports and then retransmits this data from all its other enabled ports. Since the nodes are circuit-switched, the incoming data is not buffered. Hence, the network does not suffer from the

transmission delays associated with packet switched networks. The nodes provide a richness of spare interconnections which can be brought into service after a hardware fault or damage event occurs.

Why Required:

I/O Net

I/O Network Growth/Repair

IC Network Growth/Repair

InterComputer Net

Low I/O Net Transport Delay

Rule: I/O Sequencer

The Input Output Sequencer (IOS) is an autonomous asynchronous interface between an AIPS Fault Tolerant Processor (FTP) and an I/O network. It resides on the shared bus of the FTP and can be accessed by either the Computational Processor (CP) or the I/O Processor (IOP). A major function of the IOS is to carry out detailed communication with I/O devices on the network as well as the network nodes, off-loading the FTP from lower level I/O functions.

An IOS is capable of running chains of transactions on an I/O network and collecting data from sensors at the same time as the I/O Processor (IOP) is performing its various functions, thus decoupling I/O activity from processor activity.

If the system is configured with parallel networks, the processor can be processing data from one network while a network repair chain is regrowing the faulty network.

Because it is a hardware device independent of and faster than the relatively slow IOP, the IOS can interface to very high speed buses.

Each channel of a redundant FTP may contain an IOS. These IOSs, if connected to separate independent networks, can all be active simultaneously. However, if all of the IOSs are connected to the same I/O network, then only one should be enabled to transmit at a time.

Redundant network interfaces (i.e. root links to the same network) must have their IOSs occupy corresponding address spaces within their respective channels. This facilitates dual ported memory testing and allows modifications to chain programs and chain data to be made simultaneously to all redundant interfaces to the network.

Why Required:

Low I/O Net Transport Delay

Appendix 2.E: IC Net Specification

IC Net Rules:

- Circuit Switched Nodes
- IC Interface Sequencer
- Reliable Contention Protocol
- Reliable IC Communications
- Triplex InterComputer Net

Rule: Circuit-Switched Nodes

In steady state, the communication path operates as if it were a conventional, time division multiplex bus. It differs from a linear bus in that the data is routed by circuit switched nodes along one of several possible paths. Each node in a properly configured, fault free network receives transmissions on exactly one of its enabled ports and then retransmits this data from all its other enabled ports. Since the nodes are circuit-switched, the incoming data is not buffered. Hence, the network does not suffer from the transmission delays associated with packet switched networks. The nodes provide a richness of spare interconnections which can be brought into service after a hardware fault or damage event occurs.

Why Required:

Low I/O Net Transport Delay

Rule: IC Interface Sequencer

Communication among FTPs occurs over the three-layer InterComputer Network. Access to the network is arbitrated by a distributed contention protocol. The communication and bus contention functions are performed by a dedicated InterComputer Interface Sequencer (ICIS) connected to each channel in a FTP.

Why Required:

Low IC Net Delay

Rule: Reliable Contention Protocol

The AIPS architecture consists of a three layer circuit switched bus connected in a fully cross-strapped manner to triplexes, duplexes and simplexes. FTPs, regardless of their redundancy level, compete for all three layers of the network. The contention logic must be able to resolve bus contention in a reliably robust manner. This means all FTPs must come to a consensus about who owns the bus and in addition redundant channels of a FTP must also reach a consensus as to whether or not they are contending for the bus. The bus contention protocol does not occur in the FTP channels themselves but at a dedicated ICIS controller connected to each channel in a FTP.

For access arbitration purposes, the triplex network is treated as a single entity. FTP's, regardless of their redundancy level, compete for all three layers of the network. At the end of the contention sequence one, and only one, FTP may have access to all three layers of the network. Thus, if a duplex FTP wins contention, it is given exclusive use of all three network layers even though it can broadcast on only two of the three layers. No effort is made to maximize the network bandwidth by providing simultaneous access to a duplex FTP on two layers and a simplex FTP on the third layer, for example.

In distributed systems, the contention resolution must be fair and equitable to all sites of like redundancy level. Over a period of time, for example, all triplex FTPs should have an equal chance of getting network access. Similarly, all duplexes should be served equally well by the network as should all simplexes. However, the arbitration scheme should also be flexible enough so that a low criticality function that may be assigned to run on a triplex FTP should not hog the network. A triplex FTP should contend as a duplex or simplex, if the function requesting the communication is of appropriate low priority.

Redundant channels within a FTP must come to a consensus as to whether or not they are contending for the bus, and at the end of the contention sequence whether or not they have won access to the bus. The bus contention protocol does not occur in the FTP processors themselves, but in the dedicated Inter-Computer Interface Sequencer (ICIS) electronics, which are appended to the FTP processors.

The AIPS contention protocol uses a modified form of the Laning Poll. It consists of two parts, the redundancy contention sequence (RCS) and the priority contention sequence (PCS). The RCS consists of 3 bits: S, T, and D (denoting Start, Triplex and Duplex, respectively); and the PCS consists of three FTP priority bits followed by six FTP ID bits. The objective of the redundancy contention sequence is to resolve contention between the different levels of redundant elements contending for the bus (i.e. triplex, duplex, and simplex). At the end of the RCS, all non-failed FTP's still contending should be of the same redundancy level. The priority contention sequence resolves contention among non-failed FTP's of the same redundancy level according to the priority and the ID bits.

Why Required:

InterComputer Net

How Achieved:

One Net Interface/Channel

Rule: Reliable IC Communications

Communication between critical functions, which are resident in triplex FTPs, must not be interrupted or corrupted by faults or lower criticality functions resident in duplex and

simplex FTPs. Triplex FTPs should be given access priority over all others. Similarly, duplex FTPs should have priority over simplex FTPs.

Why Required:

InterComputer Net

Byzantine Resilience

How Achieved:

Triplex InterComputer Net

Rule: Triplex InterComputer Net

The AIPS architecture consists of a three layer circuit switched bus connected in a fully cross-strapped manner to triplexes, duplexes and simplexes. FTPs can receive data on all three layers of the IC Network. The capability of a FTP to transmit on the network, however, depends on the FTP redundancy level. Each FTP channel is enabled to write on only one network layer. Triplex FTPs are provided the capability to transmit on all three layers, duplex FTPs on only two of the three layers, and simplex FTPs on only a single layer.

Why Required:

Reliable IC Communications

Appendix 2.F: System Services Software Specification

System Services Software Rules:

- Error Traps
- Fast_FDIR
- FTP_FDIR
- Heterogeneous Load Modules
- I/O Net FDIR
- I/O Network Manager
- IC Comm Services
- IC Network Manager
- IC Source Congruency
- Layered System Services
- Partitioned Design
- Privileged Mode
- Synchronization Software
- System FDIR
- System Services on all FTPs
- Transient Fault Tolerance

Rule: Error Traps

The AIPS FTP and System Software contains traps for abnormal operations such as illegal opcodes, access violations, etc. If an Ada exception occurs at run time and the user provides an exception handler, the handler will be executed. If no handler is provided, the application task causing the error will be purged and not rescheduled.

Why Required:

Software Fault Tolerance

Rule: Fast_FDIR

Fast_FDIR must be periodically performed. Its execution frequency impacts usable FTP throughput and FTP fault latency. Faster rates will reduce FTP fault latency but increase FTP throughput overhead; slower rates will reduce FTP throughput overhead but increase FTP fault latency. CSDL studies indicate that Fast_FDIR should be executed at the fastest application rate.

Why Required:

FTP_FDIR

Low Fault Tolerance Overhead

Rule: FTP_FDIR

FTP_FDIR has the responsibility for detecting and isolating hardware faults in the CPs, IOPs, and shared hardware. It is responsible for disabling outputs of failed channel(s) through interlock hardware. It logs all faults and reports status to the FTP status reporter. It is responsible for processor exception handling and downmoding/upmoding hardware in response to configuration commands from the system manager. It is responsible for transient hardware fault detection. This redundancy management function is transparent to the application programmer.

The reliability of the FTP is greatly enhanced if channels previously diagnosed as faulty but currently operating without faults can be brought back into the FTP configuration. How a channel is recovered depends on the type of failure, i.e. whether or not the fault has caused the channel to fall out of sync. A failure in the data exchange hardware does not desynchronize a channel, while other kinds of failures do. A FTP has recovered from a fault, therefore, when the failed channel can be resynchronized, or the failed channel no longer shows errors in the data exchange hardware. When a channel has been recovered, the FTP must be reconfigured so that the recovered channel participates in the FTP operation and another fault can be tolerated.

There are three main processes involved in channel recovery. *Fast_FDIR* distinguishes between transient and hard failures when a channel recovery is being attempted in order to balance competing system needs. *Lost_Soul_Sync* is responsible for resynchronizing an unsynchronized channel, i.e. synchronizing it to the instruction level and making its internal state the same as the duplex processors. Finally the *Restart* process is invoked when a second fault or a common mode failure occurs. These faults result in a fail-safe condition, which the AIPS FTP responds to with a system restart.

Why Required:

- Common Mode Fault Tolerance
- Diagnosability
- Hardware/Software Partition
- Latent Fault Detection
- RM Independent of Application
- Software Fault Tolerance
- Transient Fault Tolerance

How Achieved:

Fast_FDIR

Rule: Heterogeneous Load Modules

The software design should treat the software for each FTP as a separate entity from the link and load point of view. That is, a change in a software module that will ex-

cute on one FTP should not necessitate a recompilation and relink of the software for another FTP.

Why Required:

Modularity

Rule: I/O Net FDIR

Failures in the network hardware can occur at any time, and it is necessary to periodically collect the status of the network nodes to verify their continued proper functioning. It is also necessary to be able to respond to communication errors detected when the network is used by I/O Communication Services to collect data for application programs. These tasks typically are scheduled to run at a much faster frequency than the network manager task. The ability to respond to these errors greatly reduces the response time for network repair.

The I/O Network repair procedure varies according to the fault. If the fault is localized to a link, the network is repaired with spare link switching. If the fault is an active node failure (babbler), then regrowth of the network is required. If the fault is a node responding out of turn, then regrowth with testing is performed. In the case of regrowth and regrowth with testing, the repair of the network may take the network offline for more than one I/O cycle. Time critical applications may need a parallel network to meet their performance and reliability requirements.

Why Required:

I/O Network Manager

Rule: I/O Network Manager

The I/O Network manager software manipulates the large number of possible interconnections between the circuit switched nodes in order to maximize the system's overall reliability and survivability. It is responsible for fault detection and identification in the network and for reconfiguration around those faults.

The network manager is responsible for the following: initial growth of a network establishing paths active to each functional node; periodic testing of each node in order to determine if the node is accessible; periodic testing of spare links in the network to ensure that they will be available when needed; and re-establishing connections to healthy nodes which are disconnected as a result of a network fault or damage. For the I/O networks, errors are detected using techniques such as time outs, illegal protocol, and cyclic redundancy codes.

The software implementation of the I/O network manager must be dynamically flexible, i.e. reconfigurable during run time. More specifically, two factors motivating the

design were the need to be able to start and stop the process on demand, and the ability to manage a network topology which is to be determined at run time. To achieve this flexibility, the I/O network manager software is data driven. This dictates that the software for the I/O network manager process consist of two major parts: a data store describing the topology of the network to be managed and the coded algorithms to provide the redundancy management. These parts would be united at run time.

Why Required:

Concurrent I/O, Computation
Diagnosability
I/O Net
I/O Network Growth/Repair
Low I/O Net Transport Delay
Low System Services Overhead

How Achieved:

I/O Net FDIR

Rule: IC Comm Services

The IC Communication Services provides local and distributed inter-function communication as a transparent service to the application user. It provides synchronous and asynchronous communication, performs error detection and source congruency on inputs, records and reports IC communication errors to IC Network Managers. IC communication can be done in either point-to-point or broadcast mode and is implemented in each FTP.

IC Communication Services are responsible for maintaining knowledge of the current physical location of functions and providing the necessary interfunction communication. The mapping of functions to physical locations can change in response to a function migration.

Why Required:

Function Migration
InterComputer Net
Low IC Net Delay
Low System Services Overhead

How Achieved:

IC Source Congruency

Rule: IC Network Manager

The IC Network Manager is responsible for the fault detection, isolation and reconfiguration of the network. The AIPS system consists of three identical, independent

IC network layers which operate in parallel to provide for fault masking and reliability of communication. There is one network manager for each network layer. These managers do not need to be co-resident and are responsible for detecting and isolating hardware faults in IC nodes, links and the IC interface sequencer. The network manager is also responsible for reconfiguring the network layer around any failed elements. This network manager function is transparent to all application users of the network.

As part of deriving congruent data, the source congruency algorithms also detect errors and perform some fault isolation. Faults are isolated to one or more of the following modules: the source FTP channel, the IC Network (one of three layers), the sink FTP ICIS or the channel, and the root link. For further isolation, coordination with the local FTP FDIR, the IC Network Manager and other FTP FDIR routines is required.

Why Required:

Diagnosability

IC Network Growth/Repair

InterComputer Net

Rule: IC Source Congruency

A potential performance and reliability bottleneck in distributed systems may be the intercomputer source congruency function. The channels of a receiving FTP can all diverge if they do not use congruent inputs from IC sources. In order to remove this single point failure, it is essential that all channels agree on the input data. The IC source congruency algorithms provide this function. The overhead of executing this function, especially in degraded modes, can be substantial if not properly partitioned into hardware, firmware, and software. In the AIPS FTP, this function is performed by the hardware data exchange mechanism.

As part of deriving congruent data from the IC Net for delivery to a FTP, the source congruency algorithms also detect errors and perform some fault isolation. Faults are isolated to one or more of the following modules: the source FTP channel, the IC Network (one of three layers), the sink FTP ICIS or the channel, and the root link. For further isolation, coordination with the local FTP FDIR, the IC Network Manager and other FTP FDIR routines is required.

Why Required:

IC Comm Services

Rule: Layered System Services

The AIPS system software is organized as a series of layers, where each layer provides services to the layers above through well-defined interfaces. Each layer is allowed to use only services provided by lower layers in the hierarchy. This means that if

an interface is changed, it affects only the layers above. Adherence to this rule aids in integration, testing, and exception handling. Exceptions are handled in the layer in which they occur unless this becomes impossible, in which case the problem is passed to higher layers for resolution.

System services are layered so that application functions are not aware of the multiplicity of processors, allowing expansion of processing capability without altering application software. Since each layer hides implementation details from the other layers, the impact of a change which does not alter the interface is confined to the affected layer. Each layer is allowed to use only services provided by lower layers in the hierarchy. This means that if an interface is changed, it affects only the layers above.

Why Required:

Adaptability

Conventional User Interface

Modularity

Software Testability

Rule: Partitioned Design

The software design in a FTP is partitioned such that the application software interacts only minimally with the hardware fault management and containment mechanisms resident in the various System Services building blocks.

Why Required:

Software Fault Tolerance

Rule: Privileged Mode

Task execution on an AIPS FTP can occur in either privileged or non-privileged mode.

Why Required:

Software Fault Tolerance

Rule: Synchronization Software

The synchronization software has two functions: initial synchronization and lost soul synchronization. The synchronization routine is an assembly language module that uses data exchange and fault-tolerant clock hardware to achieve tight processor synchronism. The processors execute three sets of data exchanges and the other instructions carefully timed to take a known number of Fault-Tolerant Clock (FTC) cycles. Then they delay a different number of FTC cycles based on the configuration. The CPs and IOPs must coordinate this activity since they used shared resources.

After the CPs and IOPs are in tight synchronism, each is responsible for aligning its volatile memory and control registers: memory, dedicated data exchange registers and dedicated error latches, processor registers, and interval timers. The CP is responsible for aligning the shared hardware: the shared memory, the test port memory, the system timer, the shared data exchange registers and shared error latches and the interprocessor (CP, IOP) discretes.

The duration of bringing a lost channel back is driven by the amount of volatile storage in the system. To assure identical states of redundant hardware channels, all volatile hardware control registers, volatile memory and processor cache and registers must be identical at system start up and the redundant channels must be synchronized to the instruction level.

After a transient fault or repair event, it is necessary to bring the "lost" channel back to synchronism with the others and to align all its volatile memory, hardware control registers and processor cache with the on-line channels. If a CP finds it is alone, it must notify his IOP that it is alone and then go to Lost_Soul_Sync to be picked up. If an IOP finds it is alone, it must notify his CP that he is alone and then go to Lost_Soul_Sync to be picked up.

Why Required:
FCR Synchrony

Rule: System_FDIR

System_FDIR is responsible for the collection of status from the IC Network Managers, the I/O Network Managers, and the Local FTP Redundancy Managers. It resolves conflicting local fault isolation decisions, isolates unresolved faults, correlates transient faults, and handles processing site failures.

Why Required:
Diagnosability

Rule: System_Services_on_all_FTPs

Each FTP has a access to a copy of the AIPS System Software Services executable code, allowing any FTP to perform any System Service function at any given time.

Why Required:
Flexible Function Allocation
Function Migration

Rule: Transient Fault Tolerance

Transients faults are more likely than permanent or intermittent faults. Clearly, it is not efficient to discard a channel because of a single error observance. Also, it is important to determine if an error is caused by a transient or an intermittent fault. Therefore, when a channel loses synchronism, the failure is initially handled in the same manner as a hard fault. The effect of the fault is masked in real time and the channel is taken off line immediately to prepare for a subsequent fault. Transient fault analysis then must determine if the fault is a hard fault, a transient fault or an intermittent fault. The AIPS approach is to maintain three state variables about each channel, the failure level, or "health" of the channel, the retry time, and the retry backoff time. The health of each channel is dynamically varied based on observed errors, increasing (health worsening) as errors occur and decaying (health getting better) as time passes with no errors. The retry time is the amount of time that transient fault detection waits before attempting to bring the failed channel back. It also increases with each unsuccessful synchronization. The retry backoff time is the amount of time between successive retry attempts, assuming the channel does not begin to function again. When the retry time becomes zero, an attempt is made to resynchronize the channel. If that attempt fails, the retry backoff time is doubled and the retry time is set to the retry backoff time. A failure is deemed "hard" and a channel is permanently taken off line if the failure level goes beyond a certain threshold or if the channel is not picked up after a specified number of attempts. If the fault is transient, the channel will come back on line and remain on line while its failure level decays. If the fault is "hard", its failure level will cross the threshold or the channel will not be picked up by the operating channels in the designated number of times over the designated period of time. If the fault is intermittent and occurs frequently enough, it is treated as a hard fault and the channel is taken off line. If it does not occur frequently enough, it will be mis-diagnosed as a transient and the hardware will be brought back on line.

Why Required:

Mission Reliability

How Achieved:

FTP_FDIR

Appendix 2.G: Guidelines

List of Guidelines:

- Ada Language**
- Contention for Shared I/O**
- Copies in only $2f+1$ FCRs**
- Extra Transactions**
- Function Prioritization**
- Hardware/Software Partition**
- I/O Crosstrapping**
- Latent Fault Detection**
- Multiple Connections/IOP**
- Multiple IOSs/FTP**
- N-Version Software**
- Non-BR Fault Tolerance**
- Simultaneous Device Access**
- Spare Link Cycling**
- Spare Links**
- Variable # Processors/Channel**

Guideline: Ada Language

The AIPS is programmed in the Ada language.

The AIPS software is modular. Ada helps achieve these goals with data abstractions and packages. Ada's use of data abstractions helps produce readable code by allowing programmers to manipulate data in a conceptual manner rather than a manner specified by the machine's representation of the data. Ada also helps produce modular code by encapsulating programs in constructs called packages, which introduce data abstractions.

The foundation of the system software for AIPS is a real time, multi-tasking operating system providing mechanisms for task scheduling, inter-task communication, memory management and interrupt handling.

The AIPS operating system consists of the vendor supplied Ada Run Time System (RTS) along with those extensions needed to implement the functions of task execution management, memory management, intertask communications and software exception handling. The extensions are written in Ada with time critical sections done in assembly language to reduce system overhead. The AIPS operating system resides on every IOP and CP.

The use of Ada exception handlers and Ada types in the application tasks will prevent many application software errors during run time because most software errors will be detected at compile time or during preliminary testing. If an Ada exception occurs at run time and the user provides an exception handler, the handler will be executed. If no handler is provided, the application task causing the error will be purged and not rescheduled.

The AIPS RTS is compatible with future Verdex releases with minimal changes to interface routines. Verdex 5.5 with the AIPS RTS is based upon features of both Verdex 5.4 and 5.5. The AIPS RTS Kernel is tailored to the AIPS architecture thus eliminating time consuming general purpose routines. Also, the AIPS RTS Kernel includes all AIPS System Software.

Why Required:

Software Fault Avoidance
Software Fault Tolerance
SW Development Environment
Modularity

Guideline: Contention for Shared I/O

I/O Networks which are shared among a number of FTPs are contention networks, i.e. each FTP subscriber must win a contention before utilizing the network, and the cost of the contention (in time) is not significant. Consequently, users of I/O devices on the shared buses (global and regional) must be prepared to accept some delays due to both contention over use of the bus by two or more computers, and queueing delays due to requests to use the same device concurrently (for devices that must be 'locked' for more than one bus transmission duration).

Why Required:

Low I/O Net Transport Delay

Guideline: Copies in only $2f+1$ FCRs

While the Byzantine Generals Problem requirements dictate that $3f+1$ (four for $f=1$) independent processes which are mapped to $3f+1$ independent FCRs be provided, a triplex FTP implementation has three processes and six FCRs. This is done to minimize the hardware overheads since of the six FCRs, only $2f+1$ (three) are full identical processor channels while the other three are very simple communication repeaters or interstages. These FCRs need only contain relatively simple hardware for participating in the $f+1$ -Round Source Congruency Input Distribution function. Thus the system reliability is increased by providing an interstage design with less complex hardware than a processor.

Why Required:

3f+1 FCRs

Cost Effectiveness

Guideline: Extra Transactions

The user may provide extra transactions in anticipation of the addition of new nodes to the I/O network.

Why Required:

Variable Local I/O

Guideline: Function Prioritization

By prioritizing functions according to their criticality, it is possible to sustain multiple failure and damage events and still continue to perform the most necessary functions.

Why Required:

Graceful Degradation

Guideline: Hardware/Software Partition

Implementation of the FDIR scheme is partitioned into hardware and software. The fault detection mechanisms are implemented in hardware while the isolation and reconfiguration mechanisms are implemented in software. Tight synchronization coupled with bit-for-bit comparison and voting of redundant FTP computations allows the FTP fault detection functions to be built in hardware.

For efficiency, fault detection and error masking are done in hardware and fault isolation and reconfiguration are done in software.

Why Required:

Low Fault Tolerance Overhead

How Achieved:

FTP_FDIR

Hardware Data Exchange

Guideline: I/O Crosstrapping

To support function migration, each network in the set may have corresponding connections to more than one FTP. However, during normal operation, access to this set of networks is reserved exclusively for one FTP.

Why Required:

Flexible Function Allocation

Function Migration

Guideline: Latent Fault Detection

Latent faults are those which currently exist but have not yet caused data exchange errors or a channel to become unsynchronized.

It is important to detect latent faults before an error or active manifestation of these faults occurs. Latent faults might become active at the same time as another fault, resulting in the equivalent of a simultaneous double fault which the system is not guaranteed to handle. In fact, the most probable scenario for such a double fault is when a latent fault is lurking in the fault handling hardware or memory containing the fault handling software and that hardware and software is called upon to handle another fault. Since this hardware and software is normally only activated in response to the detection of an error, without self tests a latent fault in this area would remain undetected until the occurrence of a second fault. Self tests are designed to periodically exercise this hardware and software sufficiently to weed out such latent faults before they can cause problems.

Similarly, it is necessary to uncover latent faults in the voter and error latch hardware used as the primary means of fault detection. If an error latch does not report an error when it occurs, in effect a simultaneous double fault has occurred, which may or may not cause a system failure. The voter and error latch tests exercise and test those parts of the voter hardware whose failure would not be detected during the normal course of processing. The strategy is to apply all possible inputs to voter and error latch hardware to assure that they are working correctly.

Other self tests provide for faster detection and finer resolution in the isolation of latent faults. Typical on-line self tests include ROM checksums, RAM pattern sensitivity, comparison of RAM contents across redundant channels, voter and error latch functional tests, FTP data exchange transmitter and receiver tests, etc. The memory self tests exercise all of memory (RAM and PROM) on both the shared bus and the shared bus. PROM is tested by a sum check and a comparison of sums in redundant channels. RAM is tested by pattern writing in order to discover pattern sensitive failures. RAM address lines are tested at start up by the writing the address of each location to that location and then reading back and checking those addresses after all RAM has been written. At run time the memory test compares the RAM of the redundant channels on a word by word basis for quick detection of bit failures. The fault-tolerant clock is tested at system startup, but not during run time.

Extensive self tests of memory, hardware voters and error latches, IOS, fault-tolerant clock and system timers are done at system startup. A subset of these self tests are executed on a time available basis at the lowest priority during normal system operation to detect latent faults, particularly in seldom used hardware.

Why Required:

Mission Reliability

How Achieved:

FTP_FDIR

Guideline: Multiple Connections/IOP

Although an I/O Network may not be redundant, a FTP may have more than one connection to an I/O Network through multiple IOPs.

Why Required:

Variable Local I/O

Guideline: Multiple IOSs/FTP

The number of I/O Networks which can be managed from a given processing site is bounded by the number of physical I/O interfaces (IOSs) it has. For the engineering breadboard system, this upper limit is six. In addition, a FTP channel may contain more than one IOS for redundancy.

Why Required:

Variable Local I/O

Guideline: N-Version Software

In the AIPS, software fault tolerance may be increased using an FTP with attached processors running N-Version software and a confidence voter decision algorithm. The confidence voter has been developed in order to increase the reliability of N-Version software. It uses the deterministic nature of software failures to identify coincident faults in two versions.

The FTP/AP architecture addresses both hardware faults and software faults. Software fault tolerance is provided by Attached Processors running several independently developed programs in parallel to perform critical application functions. This FTP/AP architecture solves many of the problems associated with N-version software such as longer execution times, input congruency, abort isolation, hardware/software failure, and recovery of failed versions. The confidence voter can be used to increase reliability over a simple majority voter in dealing with the effects of coincident faults.

Hardware/software fault isolation needs to be handled in any system that runs multiple versions of software, since the recovery procedure differs depending on whether the faulty component is software or hardware. Because of limited resources, it is important not to discard both the hardware and the software components. Other operational N-

version systems do not have the unique FTP/AP architecture and so, cannot isolate faults between software and hardware. As a result, they discard both.

Based on the FTP/AP architecture, an algorithm has been developed and implemented on the LaRC Quad FTP/AP to isolate faults between hardware and software. If a disagreement occurs, the version in question is scheduled on all APs. After the results of this isolation iteration are received by the FTP, the isolation algorithm does a hardware bit-for-bit vote on the results. From this data, the fault is isolated to either hardware or software.

Why Required:

Software Fault Tolerance

Guideline: Non-BR Fault Tolerance

The AIPS architectural approach does not preclude use of conventional, relatively low-coverage techniques for fault isolation and latent fault detection. The judicious use of these techniques does not reduce the Byzantine Resilience of the AIPS FTPs or IC network.

These tests may include the following:

- RAM Tests (pattern writing, scrubbing)
- PROM Tests (sum check, sum comparison)
- Shared memory scrub
- Opcode tests
- Reasonableness tests
- Error detection and correction codes
- Internal sparing (spare bit planes, etc.)

Why Required:

Availability

Software Fault Tolerance

Guideline: Simultaneous Device Access

By connecting a FTP to redundant sensors and actuators via redundant networks, simultaneous access to redundant I/O devices can be provided. This also results in a reduction of time skew between readings and an increased I/O network bandwidth. The user is also provided with an uninterrupted flow of I/O data during periods of network reconfiguration.

The use of chains and IOSs allows the application user access to redundant devices in a nearly simultaneous fashion. If a faulty network is being repaired, the application is

able to access the redundant devices without any loss of cycle time while the faulty network is being repaired.

Applications users specify I/O requests in an hierarchical manner. Initially, the desired Transactions are specified. Next, the Transactions that are sequentially executed as a unit on one network are grouped to form a Chain. Finally, the Chains are grouped into I/O Requests to allow simultaneous execution on parallel networks on an I/O service.

Chains on parallel networks can be used to allow corresponding devices on each network to be accessed at approximately the same time. The degree of simultaneity which can be achieved is determined by three factors: the rate at which the IOS samples its Interface Command Register, the amount of time required to issue a Start Chain command, and the reproducibility of the response time for corresponding external devices.

Why Required:

Low I/O Net Transport Delay

How Achieved:

One Net Interface/Channel

Guideline: Spare Link Cycling

All spare links, including root links, are routinely cycled to determine whether or not they are operating properly and can therefore be reliably called into service to reconfigure the network after a failure of some active link. Cycling spare links provides greater fault coverage than merely testing a link and then restoring the active link to service since all parts are exercised for longer periods of time.

Why Required:

Diagnosability

Guideline: Spare Links

The AIPS I/O and IC Networks have a set of spare links which allow them to be re-configured in response to fault or damage events, rendering them resistant to such failures as a broken link, a transmitter or receiver stuck high or low, babbling network element that transmits without obeying the proper protocols, or an element which responds to messages addressed to other elements.

Why Required:

I/O Net

InterComputer Net

Guideline: Variable Memory/Channel

The amount of memory resident in a channel of a FTP may be varied without affecting the AIPS core fault tolerant concepts.

Why Required:

On-Line Memory

Guideline: Variable # Processors/Channel

An AIPS FTP is composed of a number of channels, each channel executing bitwise identical instructions on bitwise identical data. Each channel may host one or more processors. These processors may be added for the purpose of increasing the throughput of the FTP, or as spares for the purpose of increasing the attrition resilience of the channel.

A typical AIPS FTP possesses two identical processors per channel, denoted the Computational Processor (CP) and the I/O Processor (IOP). As their names suggest, the former is typically intended to perform applications tasks while the latter is responsible for I/O functions. The CP and IOP communicate over the Shared Bus using a variety of inter-processor communications mechanisms. These mechanisms include the Shared Memory where one processor may store data to be read by the other, and interprocessor interrupts whereby one processor may interrupt the other.

Also resident on the Shared Bus are the FTP's I/O Sequencers which give it access to I/O Network(s) and the IC Network. Both the CP and the IOP may access this hardware, thus making the CP/IOP distinction somewhat arbitrary.

The CP and IOP in an AIPS FTP also share the FTP's interchannel data exchange hardware, such that only one of the CP or IOP may access it at a time.

In addition to hosting a CP and IOP in a channel, an AIPS FTP may contain attached processors (APs) connected to a bus interface resident on the Shared Bus. The bus currently supported is the VMEbus. The VMEbus-compatible APs may be used for increased throughput, memory, I/O, or FTP reliability.

Why Required:

Expandable Throughput

Graceful Degradation

3.0 FORMAL VERIFICATION OF INTERACTIVE CONSISTENCY

3.1 Introduction

Verification refers to the process of demonstrating that a system implementation faithfully embodies its specification. When the process is mathematically rigorous, it is called *formal verification*. Formal verification is of interest in ultrareliable systems because of its potential to increase confidence in the system's correct implementation while reducing the cost of verification through exhaustive testing. It is useful because its mathematical rigor forces the system specifier to construct a complete, self-consistent, and unambiguous statement of exactly what the system is supposed to do. Finally, it is of interest because it can provide a consistent and traceable hierarchical framework for the specification, design, implementation, and verification of complex ultrareliable digital systems.

The feasibility of formal specification and verification of digital circuitry has been demonstrated by several researchers (e.g., [30]). However, routine utilization of the technique is currently limited to formal verification experts. While the partial results presented in this section were obtained relatively easily by non-experts in formal verification, it is possible that formal specification and verification will continue to be so esoteric and time-consuming that it will be cost-effective only for those functions which are absolutely critical to the system's fault tolerance and which in addition remain relatively unchanged from implementation to implementation.

The process of interactive consistency is precisely such a function. It is absolutely central to the fault tolerance capabilities of the AIPS Fault Tolerant Processor (FTP), the CSDL Fault Tolerant Parallel Processor (FTPP), the Fault Tolerant Multiprocessor (FTMP), the Software Implemented Fault Tolerance (SIFT) computer, the Multicomputer Architecture for Fault Tolerance (MAFT), and the Fault Tolerant Processor with Fault Tolerant Shared Memory (FTP/FTSM). Each of these architectures must perform interactive consistency in one form or another, and it is one of the functions which remains relatively unchanged over successive implementations.

The goal of this effort is to (1) formally specify the concept of interactive consistency in EHDM [9], (2) develop and specify a more detailed finite-state machine model in EHDM, (3) develop a circuit-level implementation, and (4) develop mathematical mappings between each level of specification. This document presents the current status of phase (1). In the current project, only item (1) has been completed. No major problems were found or foreseen which would make the completion of steps (2), (3), or (4) unduly difficult.

3.2 Approach

Interactive Consistency (IC) denotes the problem of distributing a datum from one member of an unreliable distributed system to other members of that system. It is com-

monly performed when one channel of a redundant computer (e.g., an AIPS FTP) possesses a datum such as a sensor value and must disseminate that datum to the other channels of the redundant computer. Because it is a performance-critical function, it is implemented in dedicated hardware in the AIPS FTP.

When this data distribution process is resilient to malicious or Byzantine failures, it is called a Byzantine Resilient Interactive Consistency (BR IC) algorithm. A Byzantine Resilient IC algorithm must satisfy the Agreement and Validity conditions. Informally, the Agreement condition specifies that nonfaulty recipients of the distributed datum must agree on the datum's final value. The Validity condition specifies that, if the initial source of the datum is nonfaulty, then the value that all nonfaulty recipients agree upon is equal to that sent by the source.

A starting point for the analysis is the classic "The Byzantine Generals Problem" paper [31], where it is rigorously shown that $3m+1$ participants are required to achieve IC in the presence of m malicious failures. Rigorously proven algorithms for BR IC are then given by [31]. The approach of phase (1) of this work is to select one of these algorithms and cast it into Revised Special.

It is important to point out that there is no need to consider faulty behavior at any level of specification or implementation of the interactive consistency circuit since the highest level specification (i.e., [31]) embodies all requirements for fault tolerance. It is only necessary to show that the lower-level implementation meets the highest-level specification, from which it follows that the interactive consistency function as a whole is Byzantine Resilient.

3.3 Outline of Development

It will be shown that the theoretically demonstrable requirements for BR IC are met by an interactive consistency circuit. These requirements and a verification technique for each may be outlined as

Requirement 1: $3m+1$ participants in protocol.

Proof: We assert that this requirement can be verified by inspection.

Requirement 2: $2m+1$ inter-participant connectivity.

Proof: We assert that this requirement can be verified by inspection.

Requirement 3: Each participant resides in a separate fault containment region.

Proof: We define each FCR as an aggregate of circuitry possessing independent power, electrical isolation, clocking, and physical isolation, and

demonstrate that the architecture meets these criteria. Given that this operative definition of an FCR is believed to be sufficient, then this requirement can be verified by inspection

Requirement 4: Fault containment regions execute a synchronous protocol.

Proof: This requirement is assumed to be met by assumption of a suitable distributed Fault Tolerant Clock (FTC).

Requirement 5: The participants correctly execute a BR IC algorithm.

Proof: This requirement is met by formal expression of a BR IC algorithm and formal proof of correspondence between circuit implementation and high-level spec.

A familiar physical arrangement of fault containment regions which is potentially capable of meeting requirements 1, 2, 3, 4, and 5 is shown in Figure 3-1.

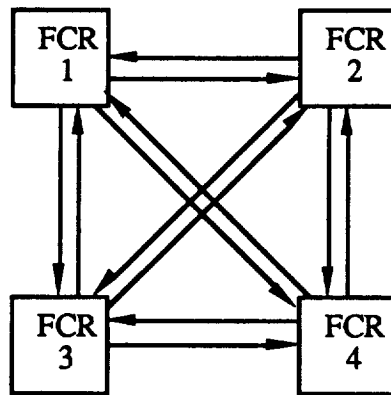


Figure 3-1. Physical Arrangement of Fault Containment Regions

A more detailed figure showing a single FCR is given in Figure 3-2. This figure is intended to illustrate how an FCR extends to the receivers of its transmissions, allowing an FCR to deliver conflicting data to different receivers without getting into the somewhat philosophical issue of whether links are FCRs unto themselves.

3.4 Assumptions

Several assumptions are made to simplify the approach.

1. The data paths between the FCRs are assumed to be 1 bit wide.
2. Synchronization is provided by an external fault tolerant clock which will not be verified at present.
3. Each participant resides in an FCR each of which possesses independent power, electrical isolation, clocking, and physical isolation.

4. The nonfaulty digital circuitry possesses zero rise and fall times, zero setup and hold times, and hence no possibility of metastability.

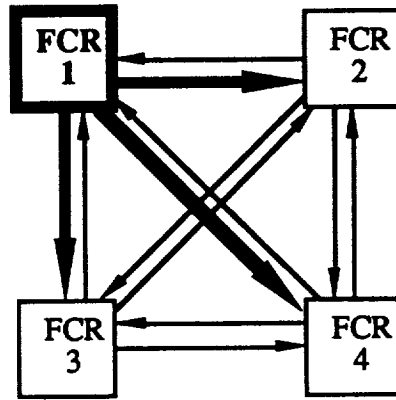


Figure 3-2. Detail of a Single Fault Containment Region

3.5 Informal High-level Specification of a Byzantine Resilient Interactive Consistency Algorithm

We choose algorithm OM(1) from [31]. This 1-BR IC algorithm is paraphrased below, for $m=1$ (# malicious faults to be tolerated at a time) and $n=4$ (# of participants in BR IC algorithm; must be $\geq 3m+1$).

1. The datum source (say 4) sends its value to every other participant.
2. For each recipient i , let v_i be the value recipient i received from 4, or else let v_i be DEFAULT if i receives no value from 4. Each recipient i then acts as a source to send v_i to each other participant (other than the original source 4)¹.
3. For each i and each $j \neq i$, let v_{ij} be the value recipient j received from source i in step 2, let v_{ij} be equal to DEFAULT if recipient j received no value from i , and let v_{jj} be equal to v_j . Each recipient j uses the value majority (v_{1j}, v_{2j}, v_{3j}) to achieve interactive consistency.

¹In practice each recipient $i \in \{1, 2, 3\}$ also delivers v_i to the original source 4, which also executes Step 3. This is a secondary technique to ensure that 4 also achieves IC in the face of source outgoing link faults. It is probably correct, but since it is not formally proven to be correct in [LSP82] it will not be specified here.

Algorithm OM makes three assumptions.

- A1. *Every message that is sent is delivered correctly.* The intent of this assumption is to prohibit an FCR from interfering with communications emanating from another FCR. An architecture with disjoint inter-FCR data paths ensures this. Note that this assumption does not prohibit a faulty FCR from sending conflicting messages to different recipients.
- A2. *The receiver of a message knows who sent it.* De-anthropomorphized, this assumption implies that the receiver of a datum can identify the sender of the datum with certainty. Satisfaction of this assumption is ensured by the physical arrangement of FCRs since each FCR possesses a dedicated port from each other FCR.
- A3. *The absence of a message can be detected.* Since we are assuming a synchronous protocol, with FCR synchrony provided by an external FTC, the absence of a message corresponds to failure of a faulty FCR to provide a datum to a recipient on the appropriate phase of the algorithm. Since the protocol is synchronous, a recipient is capable of determining that at the end of a given phase no input has been received.

The data flow for algorithm OM is illustrated in Figure 3-3. Note that in Step 2, each participant j has renamed its value v_j received on Step 1 to v_{jj} and performed a "virtual transmission" of v_j to itself. This is done for symmetry and is intended to have no effect on the algorithm or proofs of correspondence.

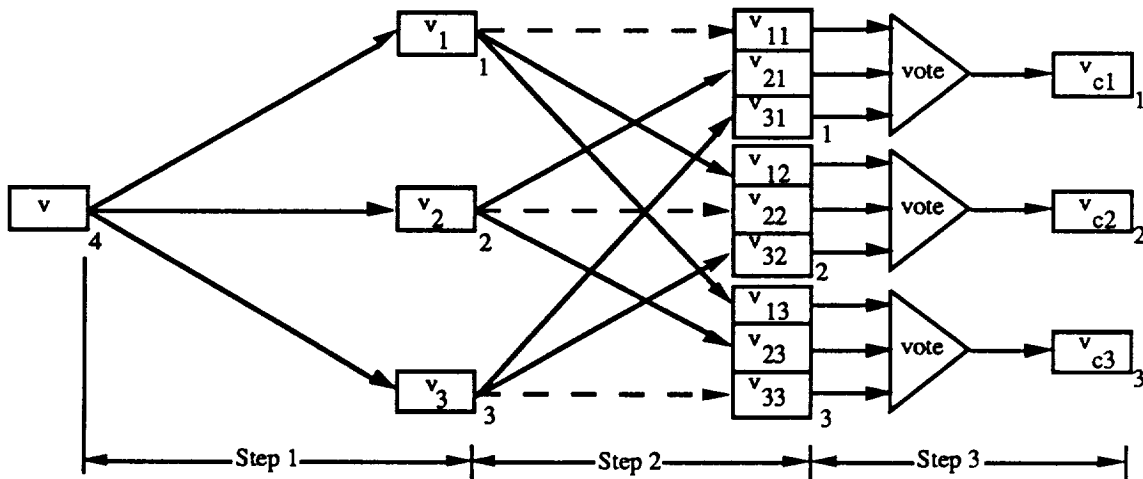


Figure 3-3. Interactive Consistency Data Flow

The task of formally verifying that Algorithm OM is correctly implemented is now reduced to proving that Steps 1, 2, and 3 are correctly implemented in the appropriate sequence. Specifically:

1. Prove that source 4 performs replication to participants 1, 2, and 3 in Step 1.
2. Prove that recipients 1, 2, and 3 of the first broadcast perform replication to recipients 1, 2, and 3 in Step 2.
3. Prove that, in Step 3, participants 1, 2, and 3 perform the majority function on the values received in Step 2.

If these functions are performed by the participants in the correct order then the circuit has been formally proven to perform interactive consistency.

A timing diagram for this algorithm is depicted in Figure 3-4.

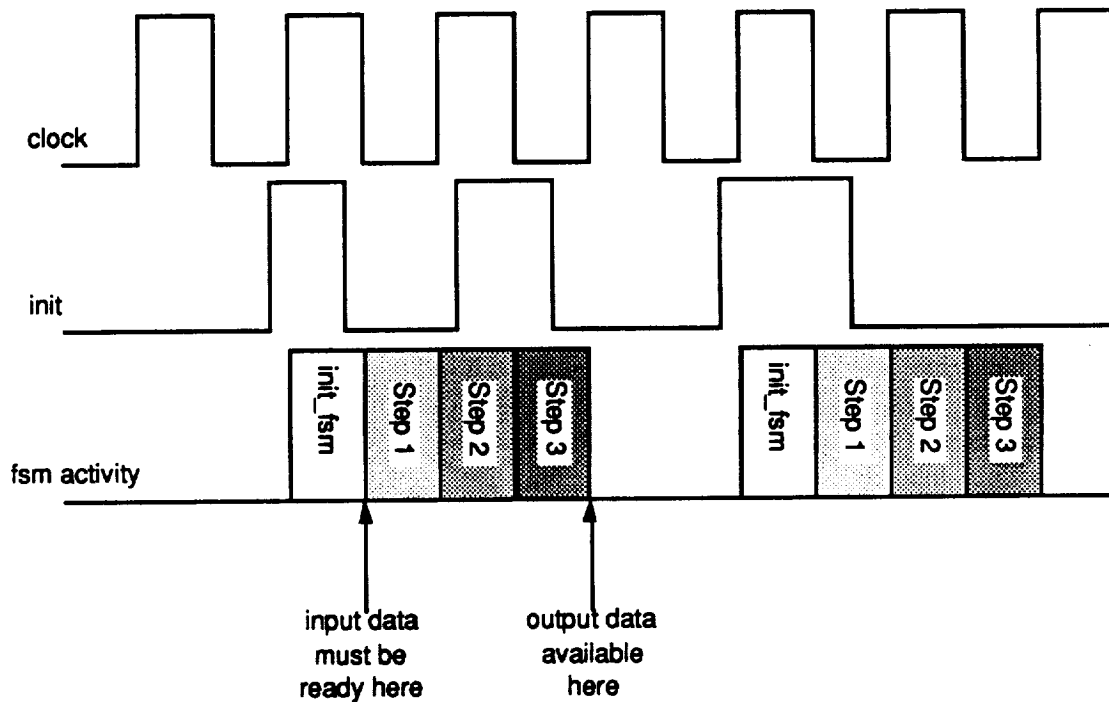


Figure 3-4. Timing Diagram for Interactive Consistency

3.6 Formal High-level Specification of Algorithm OM

This Section contains a formal specification of the Algorithm OM. The specification is written in the Revised Special language. First we perform some type definitions.

```

(*Pre-amble boilerplate*)
ic: MODULE
USING triples, quads, quints
THEORY
bool3: TYPE is triple[bool, bool, bool]
bool4: TYPE is quad[bool, bool, bool, bool]
bool9: TYPE is triple[bool3, bool3, bool3]
statevector: TYPE is quint[bool, (*the input datum*)
                           bool3, (*Step 1 results*)
                           bool9, (*Step 2 results*)
                           bool3, (*Step 3 results*)
                           bool4] (*init, Step1-3 enables*)

clock, in1: VAR bool
in3: VAR bool3
instate: VAR statevector

```

OM utilizes repeated iterations of a function which replicates an incoming data element and provides recipients with a copy of that datum. The rep function is intended to represent a faithful/restoring data replicator: it faithfully replicates the incoming data and delivers the value to its designated recipients, and it restores a meaningless input datum \notin {TRUE, FALSE} (i.e., a high-impedance or indeterminate value) to a default output \in {TRUE, FALSE} (it doesn't matter which one), which is then faithfully delivered to its designated recipients. Note that no attempt is needed nor made to specify faulty behavior of the replicator.

```

(*rep: specification for a faithful/restoring data replica-
tor*)
rep: function (bool -> bool3) ==
  (LAMBDA in1 -> bool3 :
    IF((in1 = t) or (in1 = f)) THEN
      make_triple(in1, in1, in1)
    ELSE
      make_triple(t, t, t) (*say*)
    END (*IF*)
  )(*end rep function definition*)

```

Step 3 of OM requires the use of a majority voter which votes three 1-bit inputs and delivers one 1-bit output. Note that the voter as specified can produce an output \notin {TRUE, FALSE} if a majority of the inputs \notin {TRUE, FALSE}. This is probably not the case in a real voter, which is restoring if nonfaulty.

```

(*vote3: specification for a 3-bit voter*)

```

```

vote3: function (bool3 -> bool) ==
  (LAMBDA in3 -> bool :
    IF (first(in3) = second(in3))
      THEN first(in3)
    ELSE IF (second(in3) = third(in3)).
      THEN second(in3)
    ELSE IF (third(in3) = first(in3))
      THEN third(in3)
    END (*IF*)
  ) (*end vote3 function definition*)

```

We are now ready to formally define the steps of algorithm OM. The `start_ic_fsm` function starts the state vector, if the state machine has its init signal asserted.

```

(*specification of state machine initialization*)
(*note that fsm will not execute subsequent inits until it
is returned to inittable state by Step 3*)
start_ic_fsm: function(bool, statevector -> statevector) ==
  (LAMBDA clock, instate -> statevector :
    IF( (first(fifth(statevector)) = t)
      and
      (second(fifth(statevector)) = f)
      and
      (third(fifth(statevector)) = f)
      and
      (fourth(fifth(statevector)) = f)
      and
      (clock = t))
    THEN
      make_quint(
        first(instate),
        second(instate),
        third(instate),
        fourth(instate),
        make_quad(
          f,    (*disable init*)
          t,    (*enable Step 1*)
          f,    (*disable Step 2*)
          f)    (*disable Step3*)
        ) (*end make quint*)
    ELSE
      instate

```

```

        END (*if*)
    )(*end start_ic_fsm function definition*)

(*specification of Step 1 of Algorithm OM*)
step1: function(bool, statevector -> statevector) ==
    (LAMBDA clock, instate -> statevector :
        IF( (second(fifth(statevector)) = t)
            and
            (clock = f))
        THEN
            make_quint(
                first(instate),
                rep(first(instate))
                third(instate),
                fourth(instate),
                make_quad(
                    f,    (*disable init*)
                    f,    (*disable Step 1*)
                    t,    (*enable Step 2*)
                    f)    (*disable Step3*)
                )(*end make_quint*)
        ELSE
            instate
        END (*IF*)
    )(*end step1 function definition*)

(*specification of Step 2 of Algorithm OM*)
step2: function(bool, statevector -> statevector) ==
    (LAMBDA clock, instate -> statevector :
        IF( (third(fifth(statevector)) = t)
            and
            (clock = t))
            make_quint(
                first(instate),
                second(instate)
                make_triple(
                    (
                        first(second(instate)),
                        first(rep(second(second(instate))))),
                        first(rep(third(second(instate))))
                    ),(*first element of triple*)
                    (
                        second(rep(first(second(instate))))),

```

```

        second(second(instate)),
        second(rep(third(second(instate))))
    ), (*second element of triple*)
    (
        third(rep(first(second(instate)))),
        third(rep(second(second(instate)))),
        third(second(instate))
    ) (*third element of triple*)
    ),
    fourth(instate),
    make_quad(
        f,    (*disable init*)
        f,    (*disable Step 1*)
        f,    (*disable Step 2*)
        t)    (*enable Step3*)
    ) (*end make_quint*)
ELSE
    instate
END (*if*)
) (*end step2 function definition*)

(*specification of Step 3 of Algorithm OM*)
step3: function(bool, statevector -> statevector) ==
    (LAMBDA clock, instate -> statevector :
        IF( (fourth(fifth(statevector)) = t)
            and
            (clock = f))
        THEN
            make_quint(
                first(instate),
                second(instate),
                third(instate),
                make_triple
                (
                    vote3(first(third(instate))),
                    vote3(second(third(instate))),
                    vote3(third(third(instate)))
                ),
                make_quad(
                    f,    (*disable init*)
                    f,    (*disable Step 1*)
                    f,    (*disable Step 2*)
                    f)    (*disable Step3*)
            )
        )
    )

```

```

                                )(*end make quint*)
ELSE
    instate
END (*if*)
)(*end step3 function definition*)

```

The function `ic` combines `step1`, `step2`, and `step3` to formally specify algorithm OM.

```

(*high-level specification of Algorithm OM*)
ic: function(statevector -> statevector) ==
  (LAMBDA instate -> statevector :
    step3(f,
          step2(t,
                step1(f,
                      start_ic_fsm(t, instate))))))
)(*end ic function definition*)

(*Post-amble boilerplate*)
END ic

```

3.7 Conclusions and Recommendations

This project comprises a first step in the formal specification and verification of the interactive consistency function for ultrareliable digital computing systems. The phases of a complete formal specification and verification effort are to (1) formally specify the concept of interactive consistency, (2) develop and specify a detailed finite-state machine model, (3) develop a circuit-level implementation, and (4) develop mathematical mappings between each level of specification.

The current project, which comprised only phase (1) of the overall effort, was carried out without unreasonable difficulty by non-experts in the field of formal verification. No major problems were found or foreseen in this phase which would make the completion of phases (2), (3), or (4) unduly difficult. Successful completion of these phases would result in the availability of a verified and valuable building block for the construction of ultrareliable digital systems.

4.0 AIPS FOR ALS ANALYTICAL MODELING

4.1 Introduction

Section 4 of this report describes reliability and performance analysis of the AIPS architecture as applied to the Advanced Launch System (ALS) mission. The objective of the reliability analysis is to state the reliability and availability requirements of the ALS mission, construct reliability models of the AIPS FTP and InterComputer network, construct a candidate AIPS configuration for the ALS, and evaluate its reliability and availability using the analytical models. The objective of the requirements and performance modeling effort is to determine the avionics system performance requirements for the ALS, construct performance models of the AIPS building blocks, and from these define an AIPS configuration for the ALS application. The performance is quantified using the performance models in order to gain a degree of confidence that the selected configuration of the AIPS building blocks will meet the ALS requirements.

4.2 Reliability Modeling

This section presents reliability models of the AIPS FTP and IC Network. The context of the reliability modeling effort is an assumed ALS mission scenario described below. Quantitative results are presented in Section 4.2.3.

The assumed ALS mission scenario requirements for both FTP and IC Network models are as follows:

- The ALS must reside for 1 week unattended on the launch pad
- Launch will occur only if the avionics system is known to possess the capability to mask a fault
- The ALS boost phase is of 10 minutes duration
- At least "Critical Mission Complement" (CMC) FTPs are needed to perform the ALS avionics functions
- The avionics must possess a 95% launch availability (i.e., probability of fault masking after one week of unattended operation)
- The avionics must possess at most a 10^{-5} unreliability of core avionics during the boost

The system requirement of 95% availability was taken from the ALS System Requirements Document (SRD). With these assumed ALS mission scenario requirements in mind, we have created Markov and combinatorial models for the AIPS FTP suite and the IC Network. First, a description of the FTP Markov model will be discussed followed by

a description of the IC Network Markov model. The last section will present the results obtained from the Markov models.

The quantitative results generated by the models discussed below are based on 1986-vintage components and failure rates. The ALS configuration is assumed to use 1992 technology, which should enhance reliability for a given functionality.

4.2.1 FTP Markov Model

In order to quantify the reliability analysis of the FTP, the failure modes of the system have been modeled as a series of Markov processes. In a Markov model of any system, each possible state of the system is identified. The associated state transition rates are also determined. However, before constructing the Markov model, it is necessary to identify and define the various states of the system and to compute the transition rates from one state to another. The estimation of failure rates can be made using reliability projections or in some cases by directly measuring the failure rate of similar components or modules. In order to do so, the architecture of the AIPS FTP must first be examined.

4.2.1.1 VLSI FTP Architecture

An FTP uses N modular hardware redundancy, where N is typically three or four. Three is the minimum value of N required to enable majority voting to be used to mask errors in real time, thereby eliminating the need for software rollbacks. When N is four, the effects of a second error may be masked provided the source of the first error has been previously eliminated, for example by reconfiguring the voters to ignore the input known to be faulty.

Two FTPs were modeled for the ALS: a quadruply redundant and a triply redundant FTP. The quad FTP consists of four sets (channels) of identical hardware. All channels run the same processes concurrently, exchanging and comparing data to assure each has a congruent copy. Each channel is divided into two Fault Containment Regions (FCRs): a processor and an interstage (see Figure 4-1). A FCR is a module from which no faults can propagate. Each FCR must be electrically and physically isolated from the others and provide its own independent power and clock mechanisms. The triplex FTP is identical to the quad FTP with the exception that there are only three sets of identical hardware.

The architecture of the FTP consists of a quadruply redundant Motorola 68020 processor. It contains two processors per channel with one processor devoted to computational functions and the other to Input/Output functions. The processor cards contain a Motorola 68020 32-bit microprocessor, its floating point co-processor (68882), one megabyte each of PROM and RAM memory, various other hardware elements typically found on a single board computer, and a processor specific, 6000 gate equivalent, 2.0 micron CMOS Configurable Gate Array (CGA).

VLSI FTP Fault Containment Regions (FCRs)

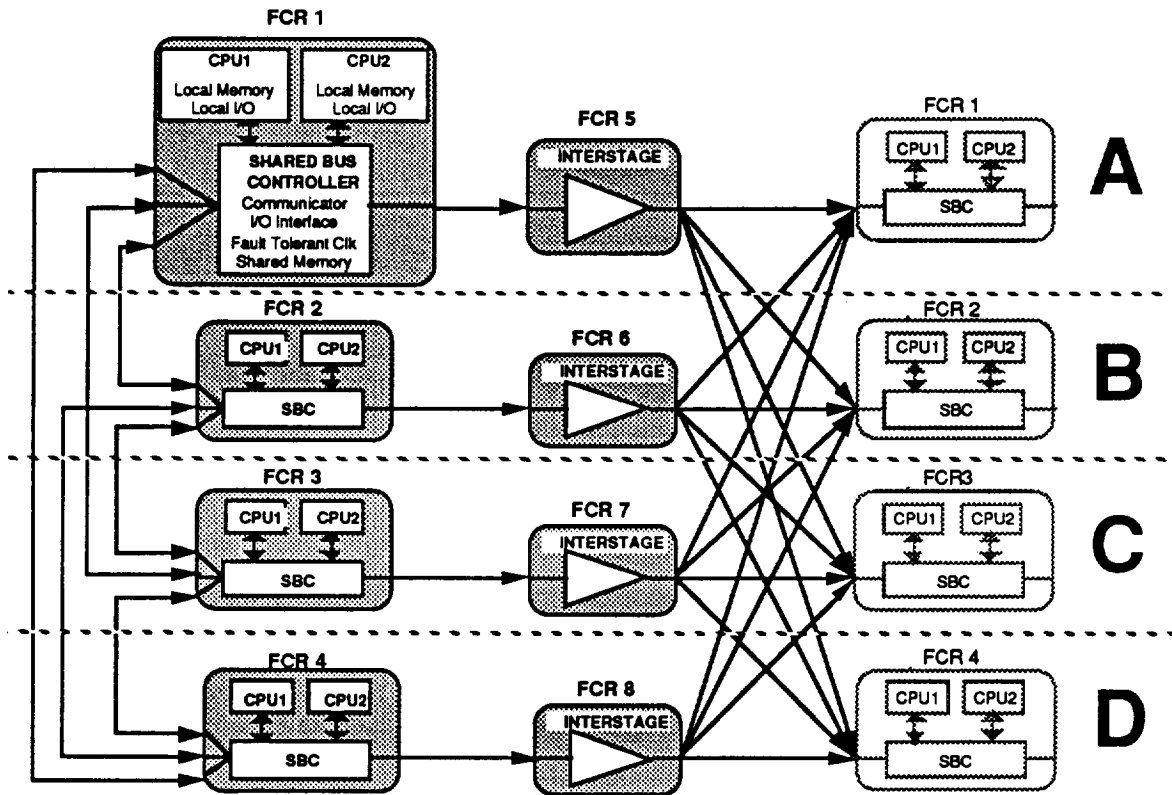


Figure 4-1. Quadruplex Fault Tolerant Processor Architecture

The AIPS FTP is resilient to arbitrary failure modes on the part of FCRs. A quad FTP provides 100% coverage of all first and second failures and an assumed 90% coverage of all third failures, i.e., the architecture is tolerant of all first and second failures no matter their severity and can incur the majority of all third failures without loss of system integrity. A triplex FTP provides 100% coverage of first failures and an assumed 90% coverage of all second failures.

4.2.1.2 FTP Failure Rates

The FTP processor and interstage failure rates were calculated by adding failure rates of the individual devices constituting the processor and interstage which, in turn, were determined from MIL-HDBK-217D. Failure rates of the processor, memory, and interstage were determined based solely on failures of solid state devices and interconnections. The data used in these calculations were gathered in the Entry Research Vehicle (ERV) study [32], in which all devices were assumed to be of Space Quality and in an airborne, uninhabited cargo environment. Failures of power supplies and backplanes

were not considered. Failure rates of the processor, memory, and interstage were based upon failure rates of the following devices:

| Device | Quantity in Processor | Quantity in Interstage | Failure Rate (per 10 ⁶ hours) |
|-----------------------|-----------------------|------------------------|--|
| Microprocessor | 2 | 0 | 0.32 |
| Floating-point CP | 2 | 0 | 0.32 |
| 64X4 static RAM | 32 | 0 | 6.0 |
| 128KX8 Bipolar ROM | 8 | 0 | 2.6 |
| VLSI Gate Array | 3 | 0 | 0.31 |
| 74xxxxxx (worst case) | 80 | 35 | 0.02 |
| PALs | 10 | 4 | 0.15 |
| Drivers/Receivers | 35 | 0 | 0.03 |
| Connectors | 4 | 1 | 0.16 |

Table 4-1. FTP Device Failure Rates ($\pi_e = 3.0$, $\pi_q = 0.5$)

Total Failure Rates

Processor Failure Rate = 28 failures/10⁶ hours. This processor failure rate does not include that of the RAM.

Memory Failure Rate = 192 failures/10⁶ hours. This memory failure rate is the product of the number of RAM devices and the RAM device failure rate.

Interstage Failure Rate = 1.5 failures/10⁶ hours.

For the ALS study, we are not assuming devices to be of Space Quality (S parts, $\pi_q = 0.5$) or resident in an airborne, uninhabited cargo environment ($\pi_e = 3.0$). We are instead assuming B quality level parts ($\pi_q = 1.0$) and a Ground, Fixed environment ($\pi_e = 2.5$) for the pad model and a Missile, Launch environment ($\pi_e = 13$) for the launch model. To reflect these changes, we have modified the above failure rates with the above multipliers for the processor, memory and interstage.

The fault recovery and reconfiguration rate, denoted by ρ , is 90,000/hr. It is assumed that fault recovery requires a mean time of one FTP frame. This is a fairly accurate estimate of the time between unambiguous fault manifestation and the setting of the vote and clock masks by the FTP System Service Fast_FDIR. For a 25 Hz frame this rate is given by:

$$1 \text{ frame} = 40\text{ms} (25 \text{ Hz.})$$

$$1/40\text{ms} * 3600\text{sec/hr} = 90,000/\text{hr.}$$

4.2.1.3 FTP Reliability Modeling Assumptions:

For the FTP reliability modeling, the following assumptions are made.

- Triplex or quad FTPs with two CPUs per channel
- Failure of either CPU takes the entire channel down
- Failure of either processor or interstage takes the entire channel down
- 1 Mbyte RAM and 1 Mbyte ROM per channel
- All FCRs are independent
- On the pad intermittent faults are assumed to be permanent faults (This assumption implies correct discrimination between transient, intermittent, and permanent faults on the part of the FTP System Services. The large amount of time available on the pad for fault diagnosis supports this assumption.)
- No recovery from transient faults is attempted during boost, that is, transients are treated by the FTP as permanent faults

$$\lambda_p = \lambda_p + \lambda_t$$

- Transient faults occur 10 times more frequently than permanent faults

$$\lambda_t = 10 * \lambda_p$$

- Failure rates are constant
- Common mode failure rate is zero
- The Critical Mission Complement (CMC) varies from one to four
- The mission starts with CMC FTPs
- Processor and memory failures are independent
- Processor base failure rate $\lambda_p = 28e^{-6}$

$$(\pi_e = 3.0; S \text{ parts}, \pi_q = 0.5)$$

- Memory (RAM) base failure rate $\lambda_m = 192e^{-6}$

$$(\pi_e = 3.0; S \text{ parts}, \pi_q = 0.5)$$

- Interstage base failure rate $\lambda_i = 1.5e^{-6}$

$$(\pi_e = 3.0; S \text{ parts}, \pi_q = 0.5)$$

- Failure rate multipliers

pad: $\pi_e = 2.5$ (Ground, Fixed); B parts, $\pi_q = 1.0$

boost: $\pi_e = 13$ (Missile, Launch); B parts, $\pi_q = 1.0$

- Constant reconfiguration rate $\rho = 25$ Hz for both permanent and transient faults
- Duplex coverage = 0.90
- Near-coincident faults result in catastrophic failure

As suggested at the "kick-off" meeting in November, 1988, both permanent faults and transient faults are addressed separately. For this analysis, it is assumed that transient faults occur 10 times more frequently than permanent faults; therefore, $\lambda_t = 10 * \lambda_p$.

4.2.1.4 AIPS FTP Markov Model

The states used for the FTP Markov model are as follows (refer to Figure 4-2). The symbols used in the model are defined in Table 4-2. The failure rates used for the pad and launch calculations are presented in Tables 4-3 and 4-4.

State 1 is a fully operational quad FTP, state 10 is a fully operational triplex FTP, state 17 is a degraded but operational duplex FTP, state 24 is a triply-degraded simplex FTP and only operational because of successful coverage by the FDIR, and state 25 is a catastrophic failure mode (two failures before recovery, four failures, etc) and is not an operational mode. States 1-10 are fault masking states. However, the system is not in a robust state while in states 2-9. While in these states, the system is trying to recover from a fault and is subject to failure due to near-coincident faults. Because of this, the system is assumed to be launched only if known to be in fault masking states 1 (quad) or 10 (triplex).

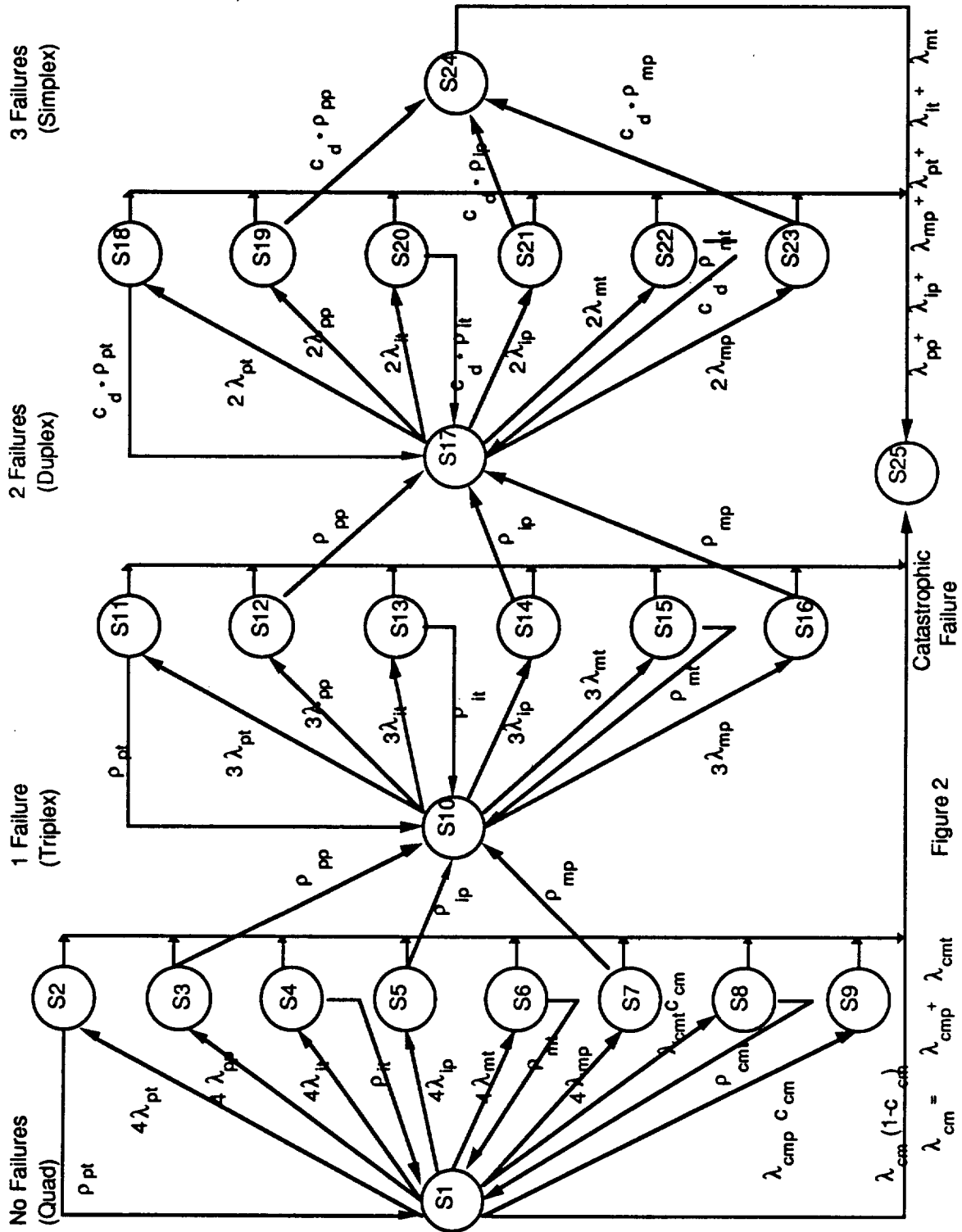


Figure 4-2. FTP Markov Model

| | |
|---|--|
| λ_{pp} = Perm. Processor Failure Rate | ρ_{pp} = Perm. Processor Reconfiguration Rate |
| λ_{pt} = Trans. Processor Failure Rate | ρ_{pt} = Trans. Processor Reconfiguration Rate |
| λ_{ip} = Perm. Interstage Failure Rate | ρ_{ip} = Perm. Interstage Reconfiguration Rate |
| λ_{it} = Trans. Interstage Failure Rate | ρ_{it} = Trans. Interstage Reconfiguration Rate |
| λ_{mp} = Perm. Memory Failure Rate | ρ_{mp} = Perm. Memory Reconfiguration Rate |
| λ_{mt} = Trans. Memory Failure Rate | ρ_{mt} = Trans. Memory Reconfiguration Rate |
| λ_{cmp} = Perm. Common Mode Failure Rate | ρ_{cmt} = Trans. Common Mode Reconfiguration Rate |
| λ_{cmt} = Trans. Common Mode Failure Rate | c_d = Duplex Coverage |
| λ_{cm} = Common Mode Failure Rate | c_{cm} = Common Mode Coverage |

Table 4-2. Definition of Symbols

| PAD | | |
|-----------------|------------|---------------------|
| | (per hour) | |
| λ_{pp} | 46.67e-6 | ρ_{pp} = 9.0e4 |
| λ_{pt} | 46.67e-5 | ρ_{pt} = 9.0e4 |
| λ_{ip} | 2.5e-6 | ρ_{ip} = 9.0e4 |
| λ_{it} | 2.5e-5 | ρ_{it} = 9.0e4 |
| λ_{mp} | 3.2e-4 | ρ_{mp} = 9.0e4 |
| λ_{mt} | 3.2e-3 | ρ_{mt} = 9.0e4 |
| λ_{cmp} | 0 | ρ_{cmt} = 60 |
| λ_{cmt} | 0 | c_d = 0.90 |
| λ_{cm} | 0 | c_{cm} = 0 |

Table 4-3. Pad Failure Rates

| | | |
|---------------------------|------------|---------------------|
| | LAUNCH | |
| $\lambda_{pp} = 2.666e-3$ | (per hour) | $\rho_{pp} = 9.0e4$ |
| $\lambda_{pt} = 0$ | | $\rho_{pt} = 0$ |
| $\lambda_{ip} = 1.43e-4$ | | $\rho_{ip} = 9.0e4$ |
| $\lambda_{it} = 0$ | | $\rho_{it} = 0$ |
| $\lambda_{mp} = 1.82e-2$ | | $\rho_{mp} = 9.0e4$ |
| $\lambda_{mt} = 0$ | | $\rho_{mt} = 0$ |
| $\lambda_{cmp} = 0$ | | $\rho_{cmt} = 0$ |
| $\lambda_{cmt} = 0$ | | $c_d = 0.90$ |
| $\lambda_{cm} = 0$ | | $c_{cm} = 0$ |

Table 4-4. Launch Failure Rates

4.2.1.5 Modeling Approach

The approach used for the FTP reliability modeling is described below.

- Solve Markov model of a single FTP's failure behavior for 200 hours (\approx 1 week)
- Probability that any FTP is not fault masking is

$$1 - (p_1 + p_{10})^{CMC}$$

- Use state vector at 200 hours to determine initial conditions for model of launch behavior

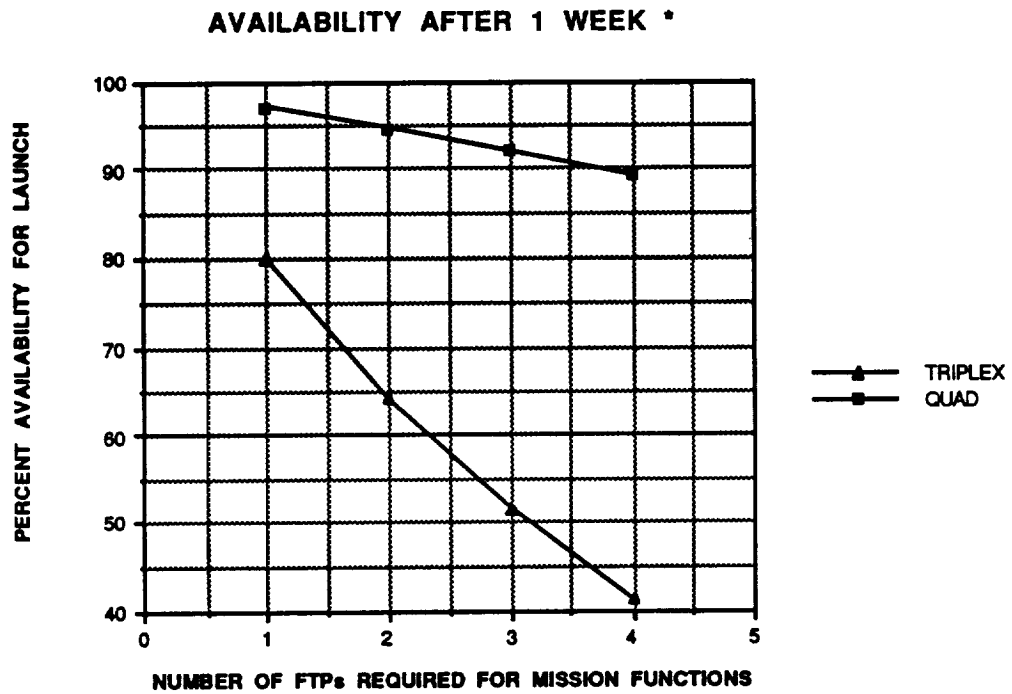
$$p_i \leftarrow p_i / (p_1 + p_{10}), i = 1, 10$$

$$p_i \leftarrow 0; \text{ otherwise}$$

- Run same model for 10 minutes at missile launch failure rates
- Probability that any FTP suffers uncovered failure during launch is

$$1 - (1 - p_{25})^{CMC} \approx CMC * p_{25}$$

The results from the reliability modeling efforts were plotted and can be seen in Figures 4-3 and 4-4. The results are discussed in greater detail later in the report.



* Assuming 1986 vintage components and failure rates - ALS configuration will use 1992

Figure 4-3. Probability of fault Masking Capability after 1 Week Unattended on Pad

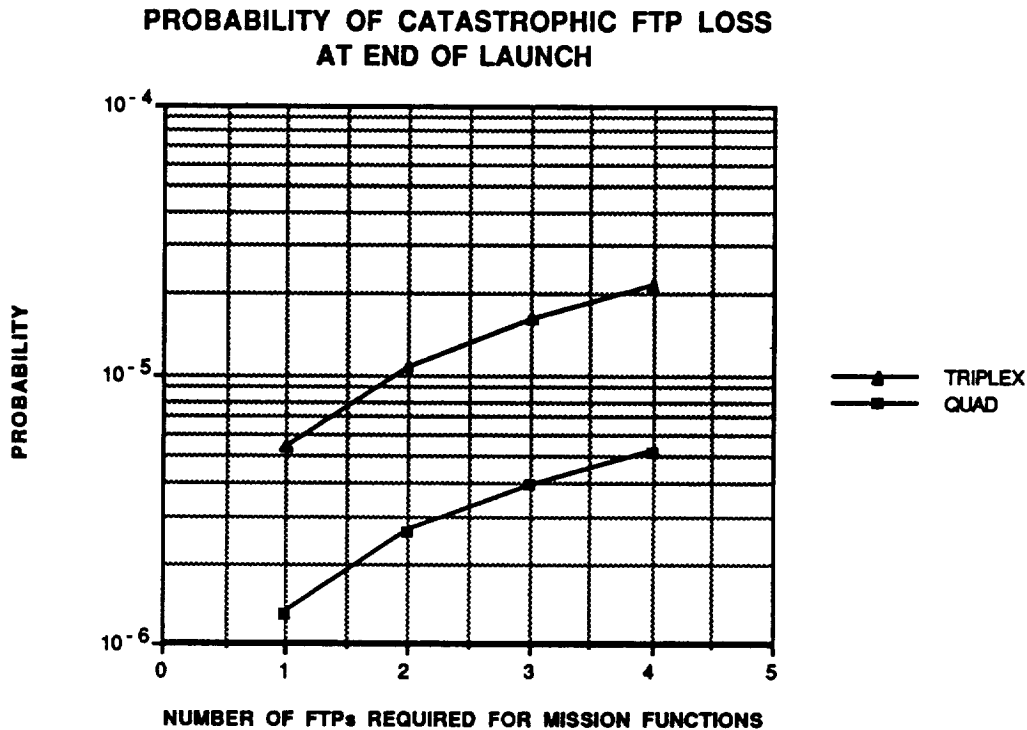


Figure 4-4. Probability of Launch Loss at End of 10-Minute Boost

4.2.2 IC Network Markov Model

The IC Network initially modeled for the ALS consists of an unreconfigurable quad redundant bus with two quad FTPs (see Figure 4-5). This IC Network is not the same IC Network used for the AIPS. The engineering breadboard AIPS IC Network consists of three identical layers of a circuit switched reconfigurable network with each layer consisting of five nodes. (See Figure 4-6). Our reason for modeling an unreconfigurable quad redundant bus with two quad FTPs was simplicity. We decided to begin our IC Network modeling efforts with two FTPs communicating with each other over a simplistic quad bus. If this configuration was not found to meet the requirements, it was intended to modify the unreconfigurable quad bus with the addition of nodes, adding reconfigurability and presumably reliability. Based on our analytical results, this was not found to be necessary.

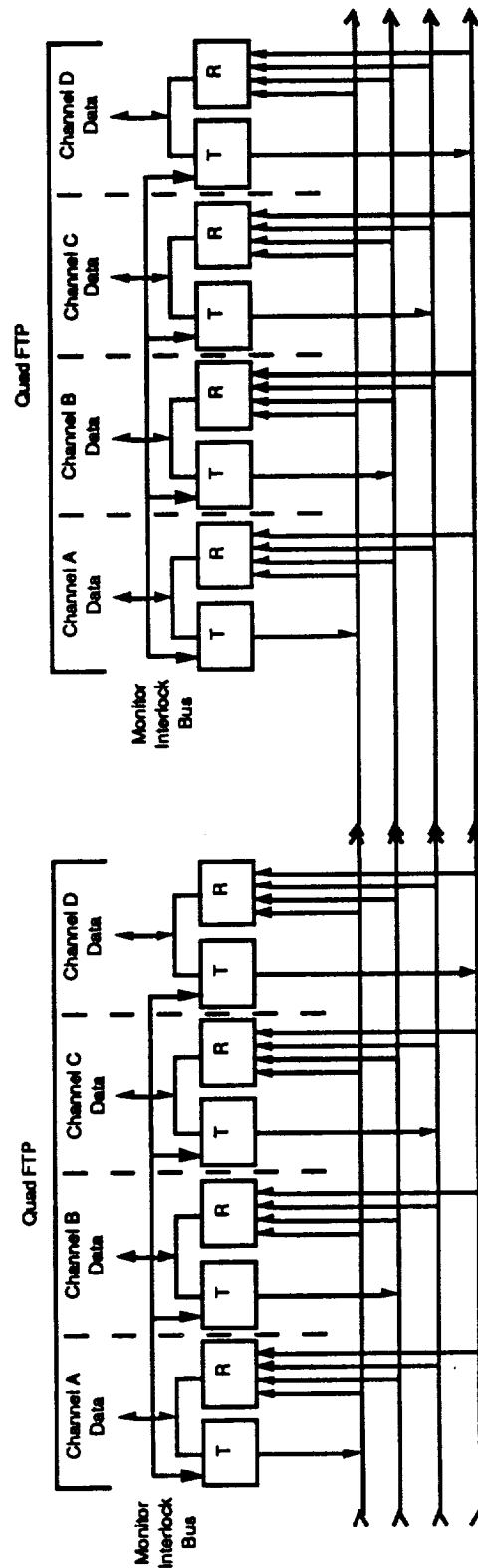


Figure 4-5. Unreconfigurable Quadruply Redundant Bus with Two Quad FTPs

AIPS ENGINEERING MODEL CONFIGURATION

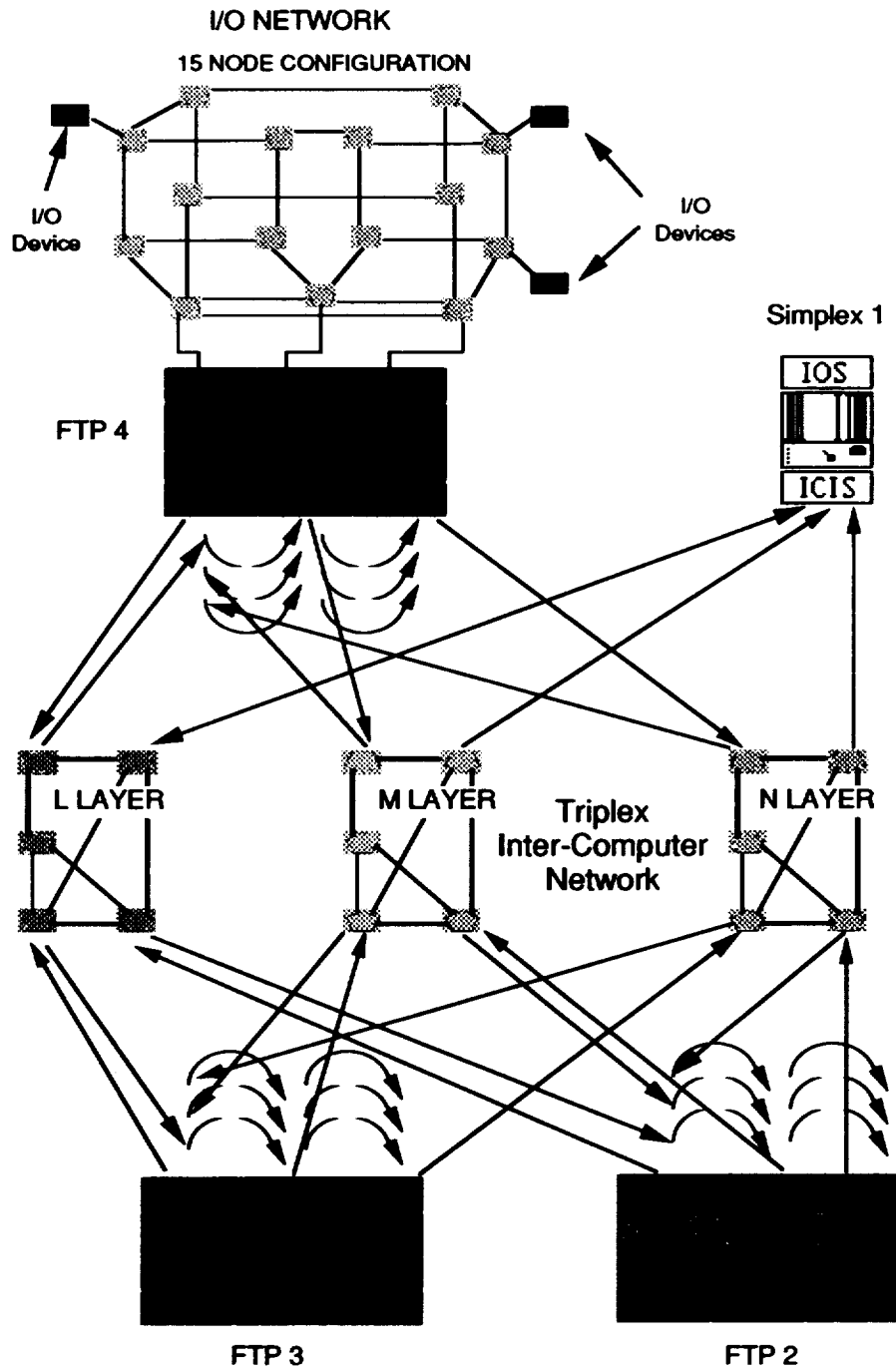


Figure 4-6. AIPS Engineering Model Configuration

The behavior of a single FTP-FTP transmission path suffering passive failures was first modeled. This meant calculating the probability that a sending FTP cannot communicate with a receiving FTP over a fault masking path. This led to the creation of the above configuration, two FTPs communicating with each other over an unreconfigurable quad redundant bus (see Figure 4-5). From Figure 4-5, one can see that the sending quad FTP has four transmitters and four receivers. This is the basis from which our Markov models were developed.

However, for an AIPS configuration consisting of more than two FTPs, it is necessary to calculate not only the probability that **one** sending FTP cannot communicate with **one** other receiving FTP but also the probability that **any** FTP cannot communicate with **any** other FTP. In order to do this, we first assumed a critical mission complement (number of FTPs) of three. Again, our reason for choosing three FTPs was for simplicity. In addition, preliminary estimates of throughput requirements for the ALS mission indicate that three FTPs are sufficient. However, we no longer can calculate the above probability from only the Markov models. In addition to the results obtained from the Markov models, we make use of a combinatorial formulation for failure of multiple inter-FTP communication paths. It is important to note that inter-FTP communication path failure probabilities are not in general mutually exclusive nor independent.

4.2.2.1 Assumptions

For the IC Network reliability modeling, the following assumptions were made.

- Transmitter, receiver, bus failures are independent with constant fault arrival rates
- Component failure rates were obtained from MIL-HDBK-217D 1553 component data, except for the Remote Terminal Interface (RTI) which was obtained from the vendor
- Failure rate multipliers are the same as used for the FTP model
- Dual failure rate (pad and launch) phased mission analysis is used, as in the FTP model
- Constant reconfiguration rate
- Transmitters are enabled via Monitor Interlocks
- Transmitters fail uncontrollably active with probability $f_a = 0.10$, passive or controllably with probability $1-f_a$
- The transmitter failure rate is given by:

$$\lambda_t = \lambda_{RTI} + \lambda_{D/R} + 2*\lambda_{conn}$$

where λ_{RTI} is the failure rate of the RTI, $\lambda_{D/R}$ is the failure rate of the driver/receiver pairs, and λ_{conn} is the failure rate of the connectors

- Receivers fail passive (a good assumption for fiber optic receivers)
- The receiver failure rate is given by:

$$\lambda_T = 4*\lambda_{RTI} + 4*\lambda_{D/R} + 8*\lambda_{conn}$$

- Duplex coverage = 0.50
- Critical Mission Complement (CMC) is 3
- On-pad recovery from transient communication faults assumed (This assumption implies successful discrimination of transient communication faults from intermittent or permanent communication faults via a retry strategy.)
- No recovery attempted for any communication faults during boost
- An unreconfigurable quad redundant bus is assumed as in Figure 4-5
- Each channel of the quad FTP can transmit data on only one layer yet receive data on all four layers

4.2.2.2 IC Network Markov Models

The approach used for the reliability modeling of the IC Network was similar to that used for the FTP. However, the Markov models created for the IC Network were developed using three separate models: one for pad phase, one for boost phase, and one for an uncontrollable active transmitter or bus failure. The latter model is applicable to both the pad and boost phases of the mission. These models are described below (refer to Figures 4-7, 4-8 and 4-9).

4.2.2.3 Pad Model

The pad model is a Markov model of two FTPs attempting to communicate over the AIPS InterComputer network. Failure modes modeled by this model are passive receiver and passive transmitter faults.

State 1 is a fully operational quad network. States 6, 7 and 20 are fully operational triplex networks. States 21-24 are degraded but operational duplex networks. They no longer have fault masking capabilities. States 1-20 are fault masking states. However, the system is not in a robust state while in states 2-5 and 8-19. While in these states, the system is trying to recover from a fault. For this reason, the system is only launched if in fault masking states 1, 6, 7 or 20. Furthermore, the pad model does not model simultaneous failures for the following reason. It is assumed that the avionics system of which the IC Network is a part has a large amount of time for fault diagnosis and recovery while on the pad. It follows that survival of simultaneous communication path failures is possible via a suitable retry protocol. Hence, pad loss will primarily be due to attrition.

4.2.2.4 Launch Model

The launch model is also of two FTPs attempting to communicate over the AIPS InterComputer network. Again, failure modes modeled by this model are passive receiver and passive transmitter faults.

State 1 is a fully operational quad network. States 6, 7 and 20 are fully operational triplex networks. States 22-25 are degraded but operational duplex networks. State 30 is a triply-degraded simplex network and operational because of good FDIR. State 31 is a catastrophic failure mode. This state is not an operational mode. States 1-20 are fault masking states. However, the system is not in a robust state while in states 2-5 and 8-19. While in these states, the system is trying to recover from a fault and is subject to simultaneous failures. It is assumed that no recovery is attempted for any communication faults during boost.

4.2.2.5 IC Bus Model

The IC Bus Markov model depicts the state of the IC Network undergoing active transmitter or connector faults. The same model is used for both the pad and launch phases; for the two phases the failure rates and initialization of the model are changed.

State 1 is a fully operational quad bus which is capable of providing fault masking operations. State 2 has suffered from one failure and is a fully operational triplex bus also capable of providing fault masking operations. State 3 has suffered two failures and is now a degraded duplex bus. In this state, the bus no longer has fault masking capabilities. State 4 is a triply-degraded simplex bus which is still operational because of good FDIR. State 5 is bus loss.

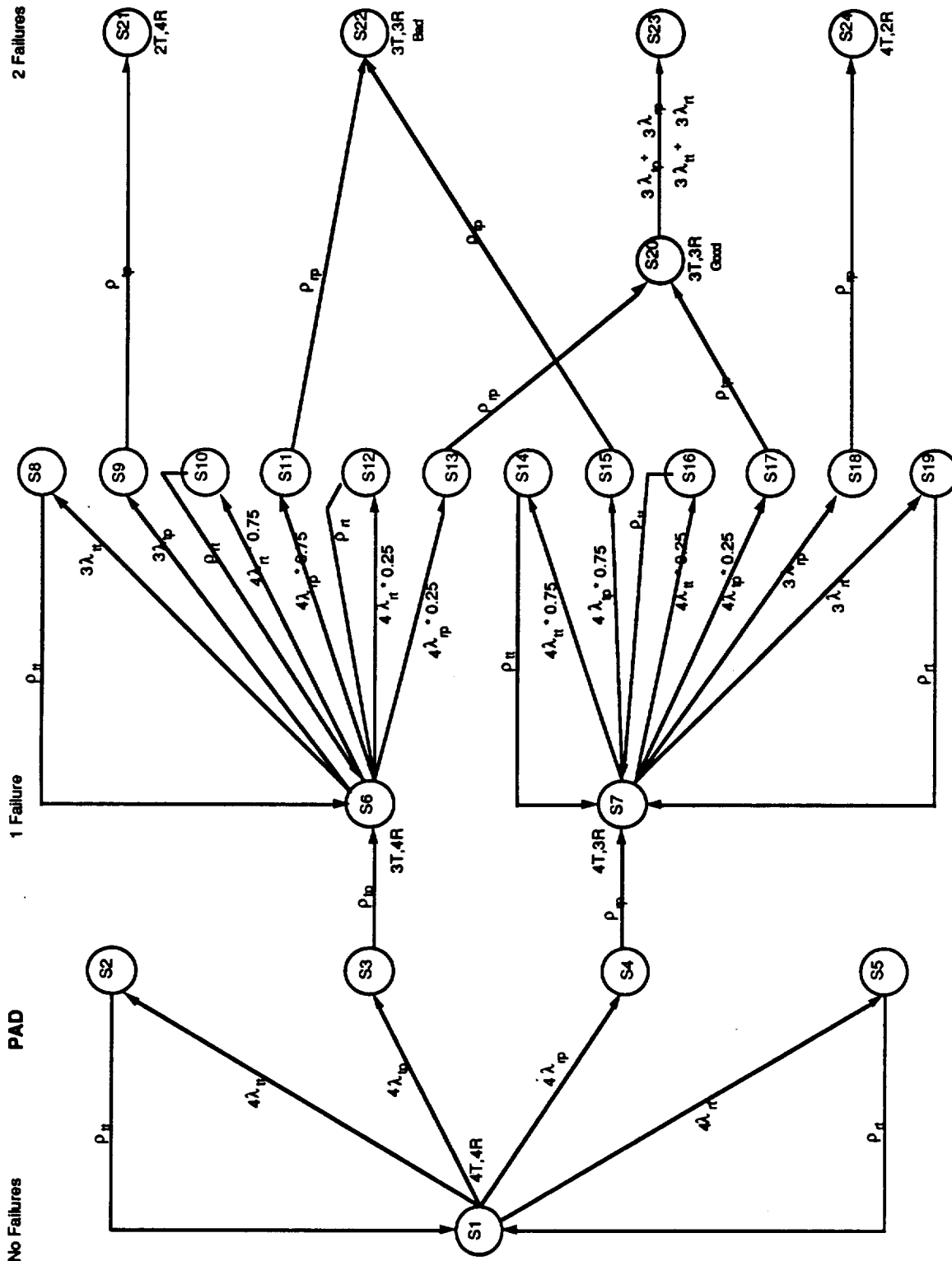


Figure 4.7. Pad Model of the InterComputer Network Transmitters and Receivers

| | |
|--|---|
| λ_{tp} = Transmitter Perm. Failure Rate | ρ_{tp} = Transmitter Perm. Reconfiguration Period |
| λ_{tt} = Transmitter Trans. Failure Rate | ρ_{tt} = Transmitter Trans. Reconfiguration Period |
| λ_{rp} = Receiver Perm. Failure Rate | ρ_{rp} = Receiver Perm. Reconfiguration Period |
| λ_{rt} = Receiver Trans. Failure Rate | ρ_{rt} = Receiver Trans. Reconfiguration Period |

PAD Failure Rates:

$$\begin{aligned}\lambda_{tp} &= 2.755\text{e-}6 \text{ failures/hour} \\ \lambda_{tt} &= 2.755\text{e-}5 \text{ failures/hour} \\ \lambda_{rp} &= 1.102\text{e-}5 \text{ failures/hour} \\ \lambda_{rt} &= 1.102\text{e-}4 \text{ failures/hour}\end{aligned}$$

PAD Reconfiguration Rates:

$$\begin{aligned}\rho_{tp} &= 1\text{e}3/\text{hour} \\ \rho_{tt} &= 1\text{e}3/\text{hour} \\ \rho_{rp} &= 1\text{e}3/\text{hour} \\ \rho_{rt} &= 1\text{e}3/\text{hour}\end{aligned}$$

Table 4-5. Pad Model Symbol Definition and Numerical Values

| | |
|--|---|
| λ_{tp} = Transmitter Perm. Failure Rate | ρ_{tp} = Transmitter Perm. Reconfiguration Period |
| λ_{tt} = Transmitter Trans. Failure Rate | ρ_{tt} = Transmitter Trans. Reconfiguration Period |
| λ_{rp} = Receiver Perm. Failure Rate | ρ_{rp} = Receiver Perm. Reconfiguration Period |
| λ_{rt} = Receiver Trans. Failure Rate | ρ_{rt} = Receiver Trans. Reconfiguration Period |

LAUNCH Failure Rates:

$$\begin{aligned}\lambda_{tp} &= 1.433\text{e-}5 \text{ failures/hour} \\ \lambda_{tt} &= 1.433\text{e-}4 \text{ failures/hour} \\ \lambda_{rp} &= 5.73\text{e-}5 \text{ failures/hour} \\ \lambda_{rt} &= 5.73\text{e-}4 \text{ failures/hour}\end{aligned}$$

LAUNCH Reconfiguration Rates:

$$\begin{aligned}\rho_{tp} &= 1\text{e}3/\text{hour} \\ \rho_{tt} &= 1\text{e}3/\text{hour} \\ \rho_{rp} &= 1\text{e}3/\text{hour} \\ \rho_{rt} &= 1\text{e}3/\text{hour}\end{aligned}$$

Table 4-6. Launch Model Symbol Definition and Numerical Values

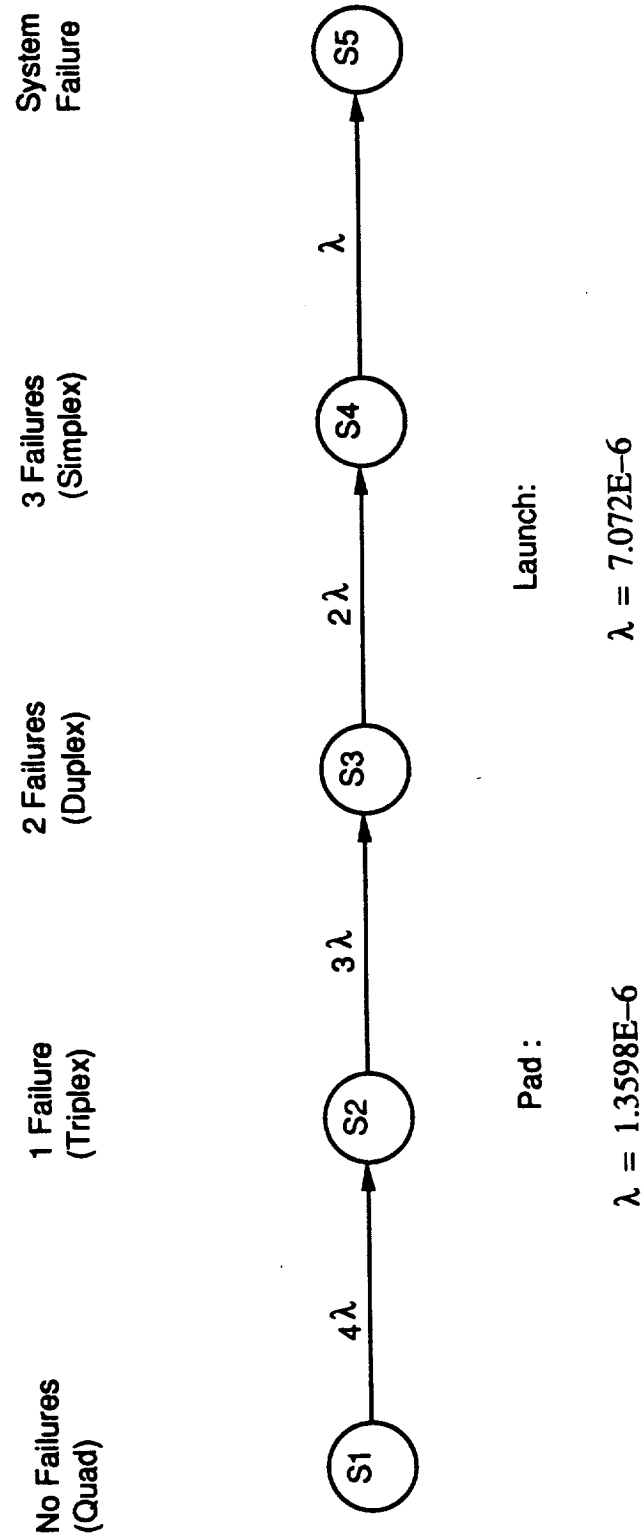


Figure 4-9. Model of the InterComputer Network: Active Failure Mode

4.2.2.6 Modeling Approach

The overall approach for reliability modeling of the pad and boost (launch) phases of the IC Network is as follows.

- Solve pad Markov model of IC Network for 200 hours (≈ 1 week)
- Use state vector at 200 hours to determine initial conditions for model of launch behavior

$$p_i \leq p_i / (p_1 + p_6 + p_7 + p_{20}), i = 1, 6, 7, 20$$

$$p_i \leq 0; \text{ otherwise}$$

- Run boost model for 10 minutes at missile launch failure rates

This approach, with some modifications, was also used for the IC Bus calculations. Below are the modifications that were made.

- Solve pad Markov model of IC Bus for 200 hours (≈ 1 week)
- Use state vector at 200 hours to determine initial conditions for model of launch behavior

$$p_i \leq p_i / (p_1 + p_2), i = 1, 2$$

$$p_i \leq 0; \text{ otherwise}$$

- Run boost model for 10 minutes at missile launch failure rates

Below is a description of the approach used for each of the three separate Markov models.

Pad Model (see Figure 4-7):

- Calculate the probability that a sending FTP cannot communicate with a receiving FTP over a fault masking path:

$$P_{21} + P_{22} + P_{23} + P_{24}$$

- Markov model of behavior of a single FTP-FTP transmission path suffering passive failures
- Perform combinatorial calculation based on CMC
- Permanent and transient faults modeled: $\lambda_t = 10 * \lambda_p$

- Only a 25% chance that a transmitter and receiver failure on the same layer will occur and result in a fault masking path

Launch Model (see Figure 4-8):

- Probability of system loss is probability that any inter-FTP communication fault is uncovered:

P31

- Expanded Markov model of pad
- 'Simultaneous' or double faults result in catastrophic failure
- Similar combinatorial formulation based on CMC

IC Bus Model (see Figure 4-9):

- Models an uncontrollable active transmitter or connector failure
- Only 10% of the failures of hardware are active (not passive) failures and result in the loss of the bus
- Failure rate for bus:

$$\lambda_{b1} = \lambda_b + \text{CMC} * \lambda_t * f_a \text{ where } \lambda_b = (\text{CMC}-1) * \lambda_{\text{conn}}$$

- Calculate the probability that the IC Bus does not provide communication over a fault masking path:

$$P_3 + P_4 + P_5$$

- Calculate the probability of IC Bus loss:

P5

- Results used for combinatorial calculations for the probability of that any FTP cannot communicate with any other FTP over a fault masking path and the probability of the IC Network launch loss based on CMC

4.2.2.7 Combinatorial Formulation

As previously stated, the reason for using a combinatorial formulation is to calculate the probability that any FTP cannot communicate with any other FTP. For our analysis, we are assuming a CMC of three. The three FTPs will be referred to as a, b and c.

Let $P(IJ)$, $(I,J) \in \{a, b, c\} \times \{a, b, c\}$, $I \neq J$, denote the probability that FTP I cannot send a message to FTP J over a fault masking path. Also let $P(IJ, KL)$, $(I,J,K,L) \in \{a, b, c\} \times \{a, b, c\} \times \{a, b, c\} \times \{a, b, c\}$, $I \neq J$, $K \neq L$, denote the probability that FTP I cannot send a message to FTP J over a fault masking path and FTP K cannot send a message to FTP L over a fault masking path. Note that $P(IJ, KL) \neq P(IJ)P(KL)$. Define $P(IJ, KL, NM)$ and $P(IJ, KL, NM, OP)$ similarly. Then p_{nfm} , the probability that any FTP cannot communicate with some other over a fault masking path, is given by

$$\begin{aligned}
 p_{nfm} = & \sum_{I, J} P(IJ) - \sum_{I, J, K, L} P(IJ, KL) \\
 & + \sum_{I, J, K, L, M, N} P(IJ, KL, MN) \\
 & - \sum_{I, J, K, L, M, N, O, P} P(IJ, KL, MN, OP) \\
 & + \text{fifth order terms and higher}
 \end{aligned}$$

The pairwise terms $P(IJ)$ are given directly from the Markov models. The quadwise terms $P(IJ, KL, NM, OP)$ may be neglected in an upperbound calculation; they will be on the order of $P(IJ)^4$ anyway. An upperbound to p_{nfm} to fifth order is therefore

$$p_{nfm} < \sum_{I, J} P(IJ) + \sum_{I, J, K, L, M, N} P(IJ, KL, MN)$$

A formulation for the triadwise terms $\sum P(IJ, KL, NM)$ in terms of the Markov model outputs is desired. An upperbound to this quantity is determined as follows.

Figure 4-10 depicts the three FTPs as vertices in a directed graph. The edges of the directed graph indicate that the FTP at the tail of the edge cannot send a message to the FTP at the head of the edge. Thus, Figure 4.10 illustrates a configuration in which no FTP can communicate with any other over fault masking paths.

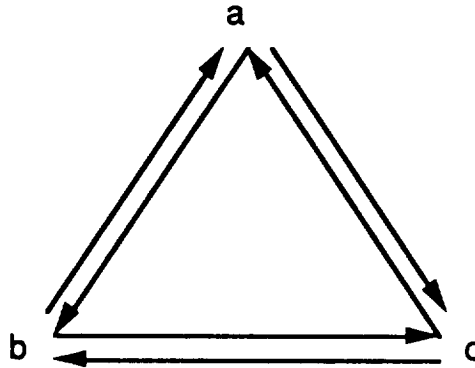


Figure 4-10. Inter-FTP Communication Paths for CMC=3

With a CMC equal to three, there are $\text{choose}(2*\text{CMC}, 3)^{\dagger} = 20$ ways that all three FTPs cannot communicate over fault masking paths (see Figure 4-11). From these twenty triadwise loss configurations, a formulation for the probability that any two FTPs cannot communicate over a fault masking path, plus the probability that all three FTPs cannot communicate over a fault masking path is

$$P_{\text{nfm}} < P(\text{ab}) + P(\text{ac}) + P(\text{ba}) + P(\text{bc}) + P(\text{ca}) + P(\text{cb})$$

$$+ \text{choose}(2*\text{CMC}, 3)P(\text{IJ}, \text{KL}, \text{MN})$$

or

$$P_{\text{nfm}} < 2*\text{CMC}*P_{\text{ab}} + \text{choose}(2*\text{CMC}, 3)*P_{\text{ab}}^2$$

where P_{ab} is the probability that a FTP cannot communicate with another over a fault masking path.

In the above expression representing the summation of the triadwise joint terms, the P_{ab} term is squared and not cubed. It is fair to assume that P_{ab} should be cubed based on the fact of the three probabilities for each combination. There are two reasons why P_{ab} is squared and not cubed in this formulation. First, we are interested in an upperbound and believe that we will achieve this limit by using P_{ab}^2 . Secondly, two of the probabilities are usually dependent with the third probability being independent. Due to the fact that two of the probabilities are dependent we can assume that since one will or won't happen, the other will or won't happen with unity probability. This leads to representing the two dependent probabilities as P_{ab} instead of P_{ab}^2 . Now when taking account the third independent probability, we obtain the term P_{ab}^2 .

This may be clarified through an example. Refer to Diagram 18 on Figure 4-11, in which FTP a has two non-fault masking incoming paths, one from b and the other from c, and FTP c cannot receive messages over a fault masking path from FTP a. The expression for the probability of occurrence of this event is $P(\text{ba}, \text{ca}, \text{ac})$ in the notation introduced above. If the receiving end of FTP a is bad, we can assume that messages from neither FTP b nor c will arrive. While this fault scenario may be caused by independent faults in the transmission circuitry of FTPs b and c, the loss of messages transmitted from FTP b and c can also be caused by faults in the receivers of FTP a. The latter fault scenario has a higher probability of occurrence and thus, to ensure that an upper bound on this probability is obtained, it is assumed to always be the cause of this failure mode. Thus the conditional probability that FTP b cannot communicate with FTP a given that FTP c cannot communicate with FTP a is assumed to be unity. At the same time, the loss of the fault masking path from FTP a to FTP c is completely independent of faults in FTP a's receiver circuitry.

$\dagger \text{choose}(n,m) = \frac{n!}{(n-m)!m!}$

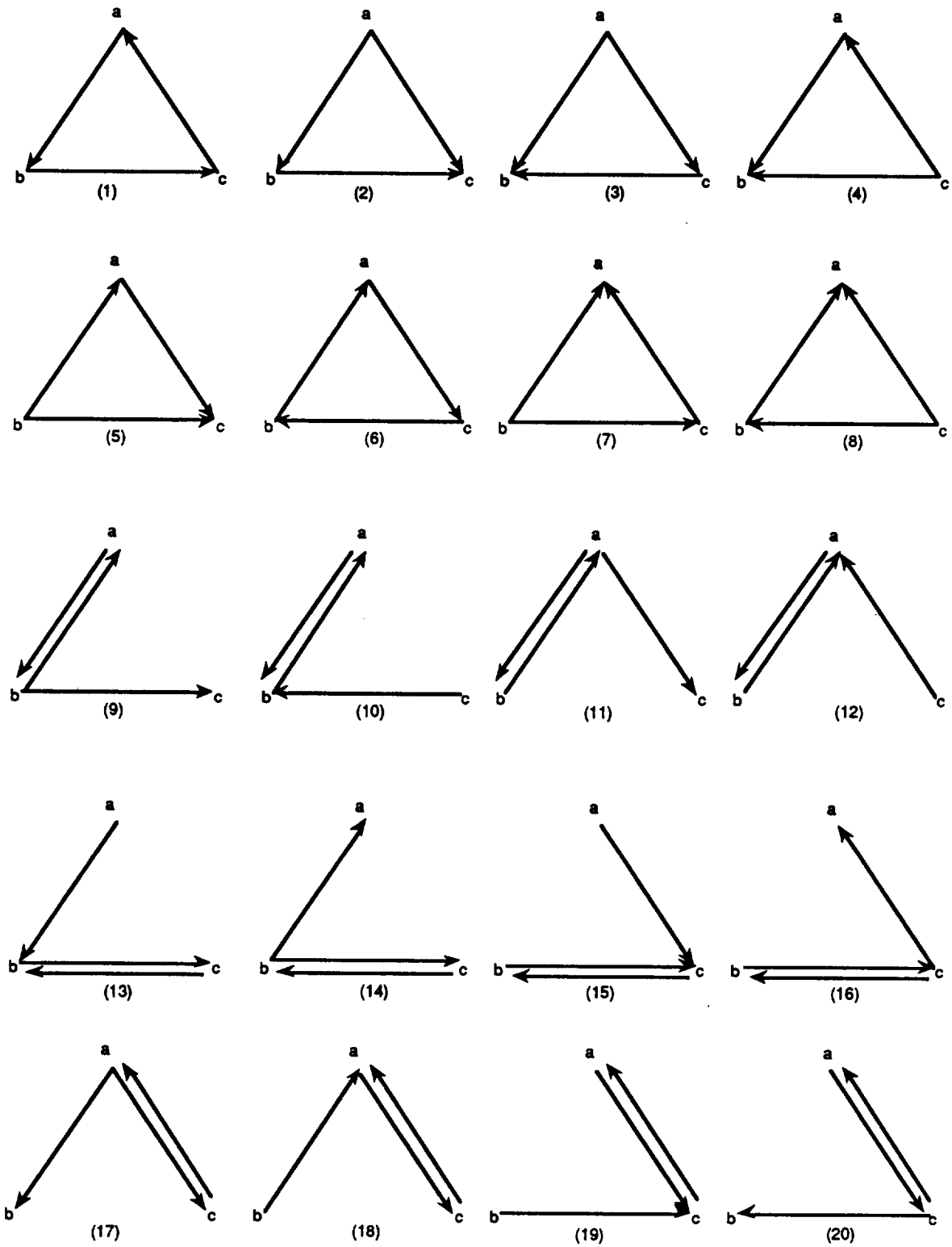


Figure 4-11. Possible Communication Loss Configurations for 3 FTPs

Therefore we write the following,

$$\begin{aligned}
 P(ba, ca, ac) &= P(ba, ca)P(ac) \\
 &= P(ba|ca)P(ca)P(ac) \\
 &< 1 * P(ca)P(ac) \\
 &< P(ca)P(ac) \\
 &< P_{ab}^2
 \end{aligned}$$

The combinatorial equation shown above only accounts for the probabilities associated with the FTP failures, and not for the probabilities associated with the IC Bus failures. The probabilities associated with the IC Bus must also be taken into consideration, which therefore leads to the following equation:

$$P_{nfm} < 2 * CMC * P_{ab} + \text{choose}(2 * CMC, 3) * P_{ab}^2 + P_{bus}$$

This equation was used to calculate the probability that some FTP cannot communicate with some other FTP over a fault masking path, and the probability that any inter-FTP communication fault is uncovered (launch loss).

The probability that some FTP cannot communicate with some other FTP over a fault masking path is calculated using the following combinatorial equation:

$$P_{nfm(pad)} < 2 * CMC * P_{ab} + \text{choose}(2 * CMC, 3) * P_{ab}^2 + P_{busnfm}$$

where P_{ab} is the probability that a FTP cannot communicate with another FTP over a fault masking path due to passive transmitter and receiver faults and P_{busnfm} is the probability that the IC Bus does not permit communication over a fault masking path due to active transmitter or connector faults.

The probability of launch loss is calculated using the following combinatorial equation:

$$P_{ll} < 2 * CMC * P_{ab} + \text{choose}(2 * CMC, 3) * P_{ab}^2 + P_{busll}$$

where P_{busll} is the probability of IC Bus loss.

4.2.3 Reliability Analysis Results

The assumed availability and reliability requirements for the ALS mission are

- 95% availability (probability of fault masking operation at launch),
- 10^{-5} maximum unreliability of core avionics at end of launch.

Another statement of the first requirement is that the probability of not possessing fault masking operation at launch (after 200 hours) must be less than 0.05. This will be referred to as P_{nfm} . The third requirement will be referred to as the probability of launch loss or P_{ll} .

The results from both the quad and triplex FTP Markov model and the IC Network model are shown below.

For the quad FTP, the results obtained are shown below. These results show the percent availability, probability of a not fault masking operation at launch (unavailability), P_{nfm} , and the probability of launch loss, P_{ll} , for varying levels of CMC.

| CMC | Percent Availability | P_{nfm} | P_{ll} |
|-----|----------------------|--------------|--------------|
| 1 | 97.24% | $2.76e^{-2}$ | $1.30e^{-6}$ |
| 2 | 94.56% | $5.44e^{-2}$ | $2.60e^{-6}$ |
| 3 | 91.95% | $8.05e^{-2}$ | $3.90e^{-6}$ |
| 4 | 89.40% | $1.06e^{-1}$ | $5.20e^{-6}$ |

Table 4-7. Analytical Results for Quad FTP

The same type of results are shown below for the triplex FTP.

| CMC | Percent Availability | P_{nfm} | P_{ll} |
|-----|----------------------|--------------|--------------|
| 1 | 80.10% | $1.99e^{-1}$ | $5.38e^{-6}$ |
| 2 | 64.20% | $3.58e^{-1}$ | $1.08e^{-5}$ |
| 3 | 51.50% | $4.85e^{-1}$ | $1.61e^{-5}$ |
| 4 | 41.20% | $5.88e^{-1}$ | $2.15e^{-5}$ |

Table 4-8. Analytical Results for Triplex FTP

The above results for percent availability and probability of launch loss are presented in Figures 4-3 and 4-4.

For the IC Network, the results obtained from the combinatorial analysis are shown below. These results assume a CMC equal to 3. Shown first is the equation and the values used to calculate P_{nfm} . These values are taken from the pad models.

$$P_{nfm} < 2*CMC*P_{ab} + \text{choose}(2*CMC, 3)*P_{ab}^2 + P_{busnfm}$$

where $CMC = 3$, $P_{ab} = 4.54e^{-5}$ and $P_{busnfm} = 4.433e^{-7}$. P_{ab} is the result obtained from the pad model, $p_{21} + p_{22} + p_{23} + p_{24}$. P_{busnfm} is the result obtained from the IC Bus pad model, $p_3 + p_4 + p_5$.

Shown next is the equation and the values used to calculate P_{ll} . These values are taken from the launch models.

$$P_{ll} < 2*CMC*P_{ab} + \text{choose}(2*CMC, 3)*P_{ab}^2 + P_{busll}$$

where $CMC = 3$, $P_{ab} = 1.85e^{-9}$ and $P_{busll} = 4.0e^{-24}$. P_{ab} is the result obtained from the launch model, p_{31} . P_{busll} is the result obtained from the IC Bus launch model, p_5

These equations for the IC Network produced the following results:

| | | |
|----------------------|--------------|--------------|
| Percent Availability | P_{nfm} | P_{ll} |
| 99.97% | $2.73e^{-4}$ | $1.11e^{-8}$ |

Table 4-9. Analytical Results for Quad IC Bus ($CMC = 3$)

4.2.4 Reliability Analysis Conclusions

The reliability analysis results indicate that a quad FTP and an unreconfigurable quad redundant IC Network will meet the ALS availability and reliability requirements of

- 1 week unattended on pad
- 10 minute boost phase
- 95% availability (probability of fault masking) after 1 week
- 10^{-5} probability of uncovered fault at end of boost or maximum unreliability of core avionics at end of the launch

The analysis indicates that after 1 week (200 hours) unattended on the pad that the percent availability for any CMC level of a quad FTP is above or very close to the 90% range. The percent availability for any triplex FTP is substantially lower. Based on these results, quad FTPs should be considered in the design of the ALS. For the IC Network, the percent availability after 1 week unattended on the pad is close to 100%. This by far satisfies the 95% availability requirement. Again one must realize that these results are based on 1986 technology. We can expect the reliability for a given functionality to increase with the use of 1992 technology.

We have assumed that if the FTP and IC Network are both capable of fault masking operation launch will occur. For the boost phase both models were initialized with the state vector obtained from the pad models after 200 hours of execution and executed at missile launch failure rates. Other changes have been made to the IC Network boost Markov model which have been discussed above. The boost phase was then modeled.

The results indicate that the AIPS system meets the 10^{-5} maximum unreliability of core avionics at the end of the launch for both the quad FTPs and IC Network. The triplex FTPs are marginal.

Both the FTP and IC Network reliability models are complete. They are now available for ALS mission analysis using actual ALS requirements and projected

technology for ALS. The AIPS for ALS Architecture Synthesis Report [8] discusses the ALS avionics architecture and presents the modeling results using the projected failure rates for 1992 technology.

4.3 Requirements and Performance Modeling

The objective of the requirements and performance modeling effort is to determine the avionics system performance requirements for the ALS, construct performance models of the AIPS building blocks, and from these define an AIPS configuration for the ALS application. The performance is quantified using the performance models in order to gain a degree of confidence that the selected configuration of the AIPS building blocks will meet the ALS requirements.

4.3.1 Performance Analysis Approach

The approach to the performance analysis of the AIPS for ALS is to obtain representative ALS avionics requirements from an airframe contractor, construct a performance model of the AIPS (which does not reflect the redundancy of the building blocks other than in the performance dimension), transform the user-generated requirements into requirements suitable for use in architecture synthesis, generate a mapping of the avionics functions to a configuration of the AIPS building blocks, and analyze the performance resulting from that mapping. Once the requirements are determined and converted into a suitable format, this process will be iterative, with the function allocation technique being iteratively coupled with a performance analysis model to determine an architecture and allocation which meet the function requirements.

4.3.1.1 Architecture Synthesis Inputs

The inputs to the performance-related architecture synthesis process comprise the avionics requirements, the AIPS building block knowledge base, which includes the throughputs, bandwidths, overheads of the AIPS FTPs, IC network, I/O network, system services, and technology projections for the anticipated technology freeze date.

4.3.1.2 Architecture Synthesis Outputs

The output of the process is a physical configuration of the AIPS for the ALS, along with a performance model output which quantifies the performance of the configuration when executing the ALS application. Components of this configuration include the number of FTPs in the system, their throughput and memory capabilities, the system services complement, the allocation of avionics functions to system building blocks, the degree of I/O cross strapping, the I/O and IC Network topologies and bandwidths. Quantitative estimates of system performance when executing the ALS application are also outputs of this process. These performance measures include:

- Input and output latency

- Processing lag
- Transport lag
- Jitter
- Component utilization
- Queue sizes

4.3.2 Advanced Launch System (ALS) Functions

4.3.2.1 Processing Specifications

Each task to be executed by the AIPS is represented by a specification. The parameters which constitute the specification are categorized into processing requirements, memory requirements, and input/output and interfunction communication requirements.

To perform an accurate and meaningful synthesis of the avionics system, the following requirements are needed for each dispatchable task:

- Frame rate
- Throughput (or instructions per execution)
- Throughput margin
- Processing lag
- Scheduling requirements (e.g., preemptible or nonpreemptible)
- Task execution order dependencies
- Inter-function communication requirements (bits per iteration, latency)

Given this data it is possible to construct a distributed schedule for the task suite, quantitatively perform system sizing, and determine performance parameters using the models described in Section 4.3.3.

4.3.2.2 I/O and Interfunction Communication Specifications

The application specification must contain degree detail regarding the Input/Output and Interfunction communication requirements. From these requirements, the following parameters must be determined:

- Amount (number of bits) of input data required to each Level-3 task for each frame iteration, number of bits
- Source (I/O device or producer task) of each input datum
- Maximum allowable latency and jitter for each input datum
- Amount (number of bits) of output data produced by each Level-3 task for each frame iteration

- Destination (I/O device or consumer task) of each output datum

Maximum allowable latency and jitter for each output datum

4.3.3 AIPS Performance Model

4.3.3.1 Virtual Architecture Model

Given the existence of a set of requirements having the degree of detail outlined in Section 4.3.2, a virtual architecture model of the AIPS can be constructed and used in quantifying its performance under the application's processing and communication load. The AIPS virtual architecture model used in this analysis is depicted in Figure 4-12. It consists of a number of FTPs which are interconnected over the InterComputer (IC) Bus. Each FTP may or may not be connected to local memory-mapped I/O and a regional or global I/O Network. Regional I/O Networks may or may not be interconnected to facilitate sharing of access to I/O devices for latency minimization or fault tolerance enhancement.

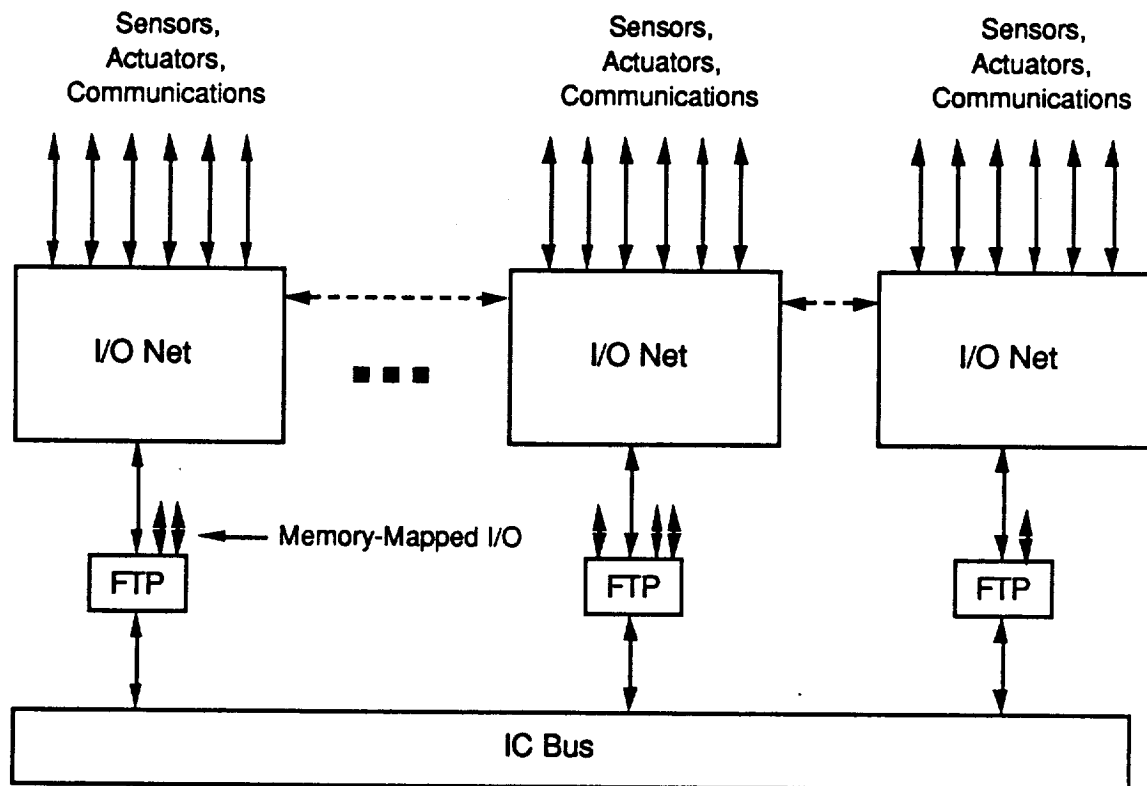


Figure 4-12. AIPS Virtual Architecture

The FTP virtual architecture used in this analysis is depicted in Figure 4-13. It is described in detail in [8].

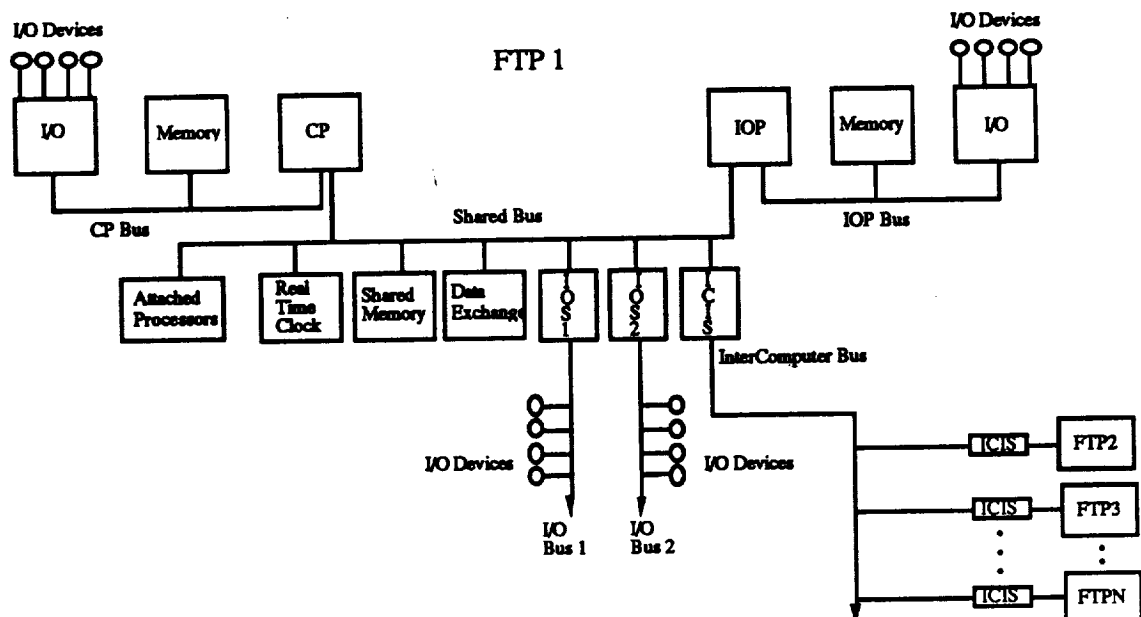


Figure 4-13. AIPS FTP Virtual Architecture

4.3.3.2 Schedule Model

We begin by expressing the behavior of a single AIPS FTP. The AIPS FTP executes an iterative, preemptive scheduling algorithm which is described below.

We assume that the FTP executes an frame-based scheduler at a 100 Hz major frame rate. This results in a major frame duration of 10 milliseconds, with minor frames consisting of time intervals of 10 millisecond multiples. Three minor frames are assumed: 50 Hz (20 millisecond frame), 25 Hz (40 millisecond frame), and 10 Hz (100 millisecond frame). Other frames periods can be easily added to the model.

The existence of these frames results in a limited number of frame types: those containing 100 Hz tasks, those containing 100 and 50 Hz tasks, those containing 100, 50, and 25 Hz tasks, those containing 100, 50, and 10 Hz tasks, and those containing 100, 50, 25, and 10 Hz tasks (Figure 4-14). It is assumed that a schedule generation tool has been used a priori to construct a static task schedule and I/O access chain for each frame type.

It is assumed that the I/O and processing are pipelined, so that on frame n the IOP prefetches input data for frame $n+1$ and transmits output data from frame $n-1$.

4.3.3.3 Frame Model

The activity of the CP and IOP during a single frame is described below and depicted in Gantt chart format in Figure 4-15. The horizontal axis of the figure represents the passage of time, while the multiple axes of the figure are labeled according to the component in use at a given time.

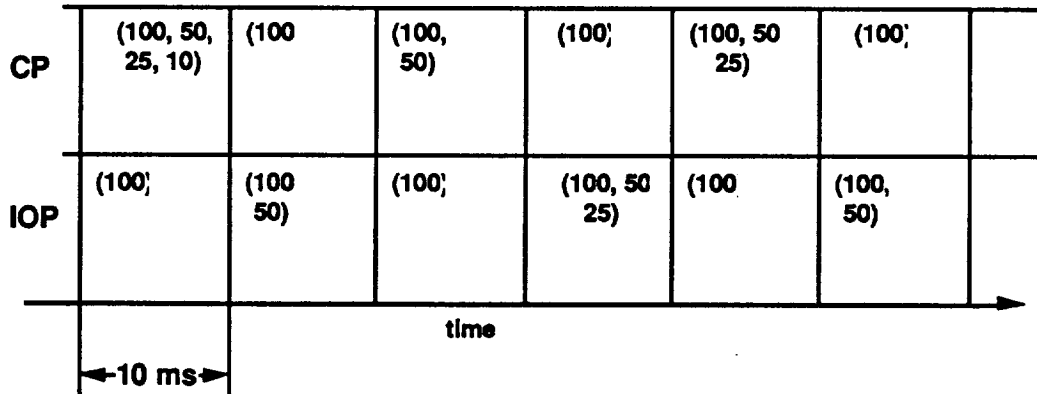


Figure 4-14. Schedule Model

On frame n, the CP executes the following sequence of operations:

- Perform FDIR
- Write output (actuator) data to Shared Memory
- Read input (sensor, status) data from Shared Memory
- Initiate frame n on IOP
- Execute task suite for frame n
- Perform self-tests
- On 10 ms interrupt, begin frame n+1

On frame n, the IOP executes the following sequence:

- Perform FDIR
- Await I/O interrupt from CP
- Read output (actuator) data from Shared Memory
- Perform I/O activity for frame n+1
- Write input (sensor, status) data to Shared Memory
- Perform self-tests
- Await frame interrupt from CP

4.3.3.4 ADAS Model

Based on these assumptions a hierarchical ADAS model of the FTP and application was generated.

Figure 4-16 depicts the model of the scheduler. It is the highest hierarchical level of the model. The node labeled "fclk" represents the frame clock. It emits a token every 10 milliseconds to the nodes labeled "delay" and "CP_fdi." The token contains an integer which indicates to the nodes of the graph and subgraphs the frame number currently being executed and hence the parameters of that frame such as the number of application tasks, the amount of data being transferred at each point, and other frame-specific parameters. CP_fdi is a function executing on the CP which is responsible for testing its own health via

a voting operation and exercising the error detection capabilities of the FTP's data exchange network. The delay node emits a token to the "IOP_fdi" node, which performs the same function for the IOP. The delay is used to prevent the CP and IOP from accessing the exchange network at the same time. The CP_fdi node emits a token to the "CP_SM" node upon completion of its tests. The "CP_SM" node models the act of reading/writing output data from the previous frame to the Shared Memory. When this process is complete, the node "SM_CP" is fed a token. The SM_CP node models the reading of input data for the current CP frame from the Shared Memory. Subsequently, the CP interrupts the IOP via passing a token to the "CP_IOP_event" node to indicate to the IOP that it may begin output activity for the current frame and input activity for the next frame. The same token is passed to the "app_tsk" node to initiate the processing of the application tasks for the current frame on the CP. Upon completion of the application tasks a token is passed to the "done" node.

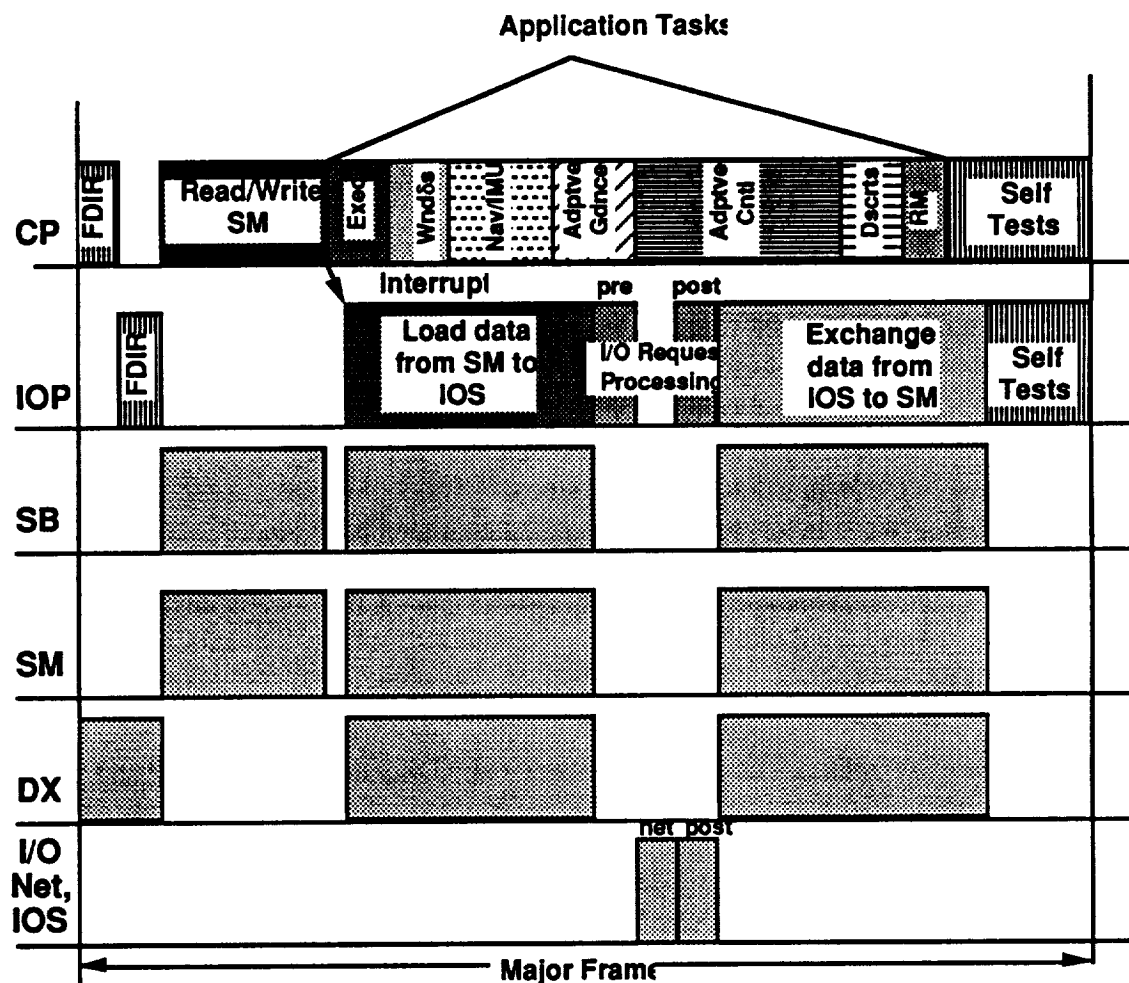


Figure 4-15. FTP Frame Activity

Upon reception of the token from the CP_IOP_event node, the IOP executes the "SM_IOS" node, which represents the reading of output data from the Shared Memory into the I/O Sequencer (IOS). Upon completion of the SM_IOS node, the "net" node is activated. This node represents the use of the IOS and the I/O Network hardware to complete the I/O chains which are designated for the current IOP frame. Following the I/O transactions, the node "IOS_SM" is activated, which represents the transfer of input data from the IOS to the Shared Memory for use by the CP in the subsequent frame. Upon completion of the data transfer, a token is passed to the "done" node.

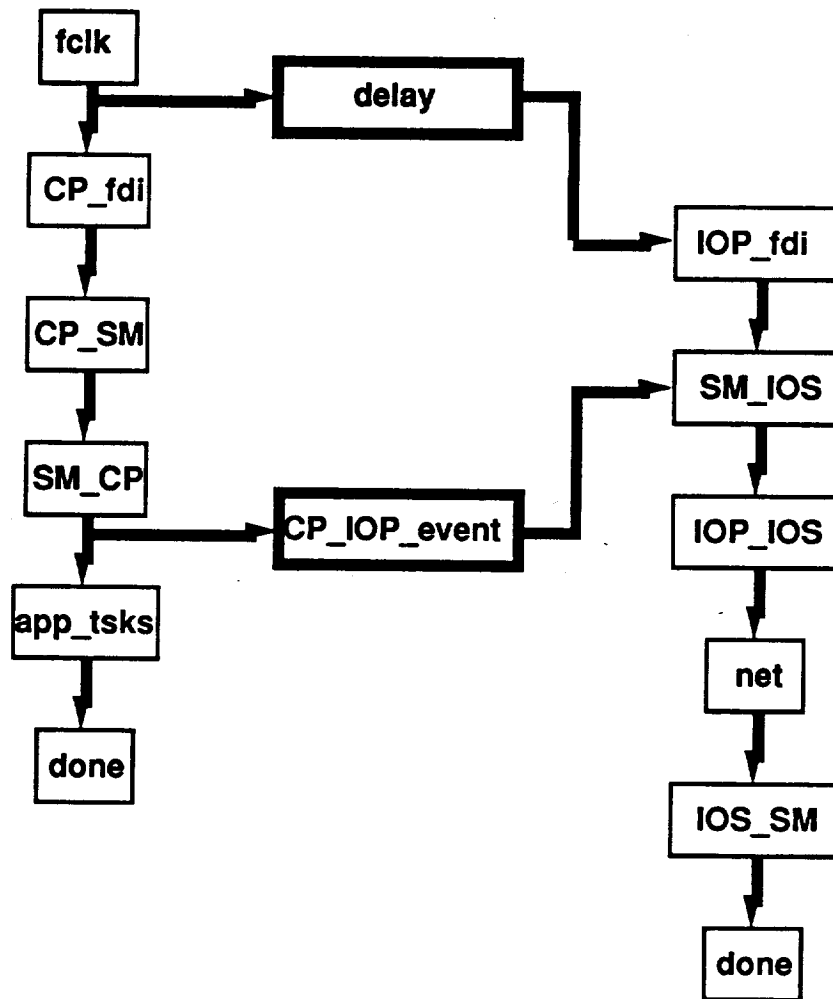


Figure 4-16. ADAS Model of Scheduler

At the next lower level of the ADAS model's hierarchy are several subgraphs. The CP_SM, the SM_CP, and the app_tsks reflect the activities of the CP (Figure 4-17). The SM_IOS (Figure 4-18) and the IOS_SM (Figure 4-19) subgraphs represent the activities of the IOP. The net subgraph (Figure 4-20) represents the behavior of the I/O network.

Three CP subgraphs were generated, corresponding to the CP_SM node, the SM_CP node, and the app_tsks node in the highest-level graph. The CP_SM subgraph

models the transfer of data from the Shared Memory to the CP. This activity requires simultaneous access to the CP, the Shared Bus (SB), and the Shared Memory (SM). The SM_CP subgraph models the reverse process, and requires identical hardware resources. Each of these graphs are provided with input data to indicate the amount of data to be transferred (determined from the frame number and the tasks to be executed in that frame), and hence the firing delay of each node in the subgraph.

The app_tsk node is provided with a token which indicates the frame number. Based on this parameter, the node labelled "sched" determines which sequence of tasks are to be executed, and iteratively selects output ports for token delivery. The sched node feeds nodes labeled "switchin," which represent the time required to perform the context switch function. The switchin nodes then deliver a token to the "task" nodes. There is one task node for each schedulable task executing on the CP; the firing delay of a particular task node corresponds to its execution time. Following completion of a task, it delivers a token to the "switchout" node, which again models the context switch time. Each switchout node delivers a token to an "any" node which delivers a token in turn to the sched node. If further tasks are to be executed in the frame, the sched node delivers a token to the appropriate switchin node. Otherwise, it delivers a token to the "out" node, returning control to the higher-level graph.

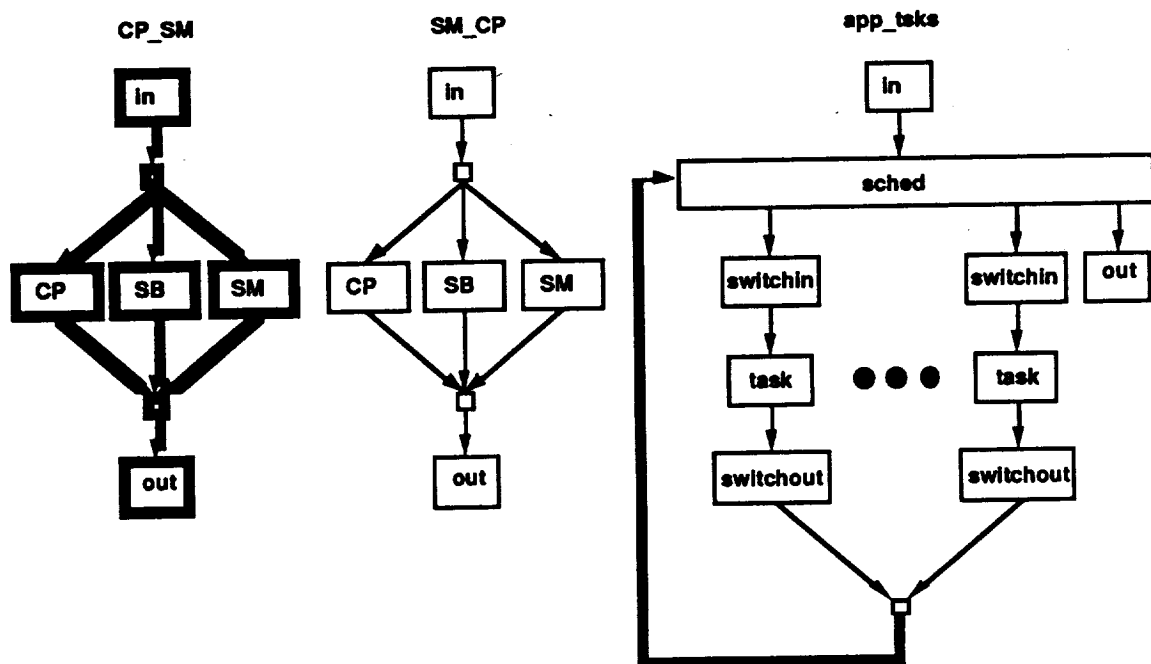


Figure 4-17. CP Subgraphs

Two IOP subgraphs were created, the "SM_IOS" subgraph and the "IOS_SM" subgraph. The former models the process of transferring I/O chain data from the Shared Memory to the IOS and the latter models the reverse process. The SM_IOS node begins with the "IO Request ID" node, which represents the process of determining which I/O re-

quests are to be processed in the current frame. Upon completion of this function, the FTP votes the I/O data from the Shared Memory through the Data Exchange (DX) network to the IOS. During this phase the Shared Memory, the Shared Bus, the IOP, the IOS, and the Data Exchange network are simultaneously in use. The token feeding these nodes contains a parameter describing the amount of data to be transferred and hence the firing delay of each node.

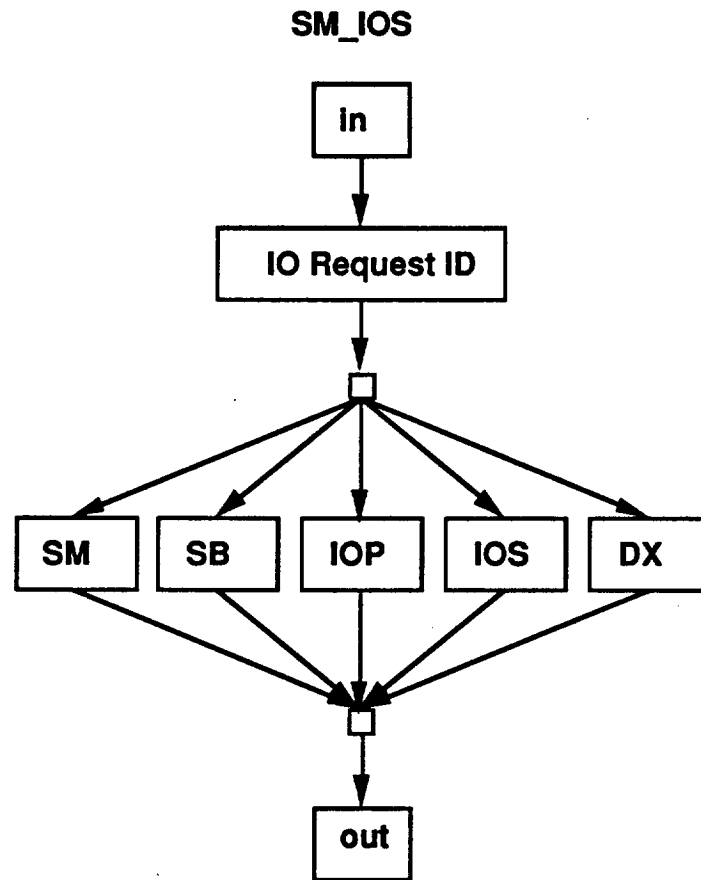


Figure 4-18. IOP Subgraphs (1)

The IOS_SM subgraph represents the process of transferring chain result data from the IOS to the Shared Memory, and consists of three sequential phases. The first phase, represented by the "IOP_process" node, models the process of the IOP performing post processing on the chain. This node delivers a token to a split which feeds the "SM," the "SB," the "IOP," the "IOS," and the "DX" nodes. These nodes model the process of transferring result data from the IOS through the voting circuitry of the Data Exchange network to the Shared Memory. The outputs of these nodes feed a join which, when all its input tokens have been received, emits tokens to a subsequent tier of nodes entitled "SM", "SB", "IOP," and "IOS." This tier of nodes represents the simultaneous use of the SM, SB, and IOP to model the process of the IOP receiving a signal from the IOS that the chain is complete. The outputs of these nodes feed a join which, when all its input tokens have been received, emits a token to the "out" node, returning control to the higher-level graph.

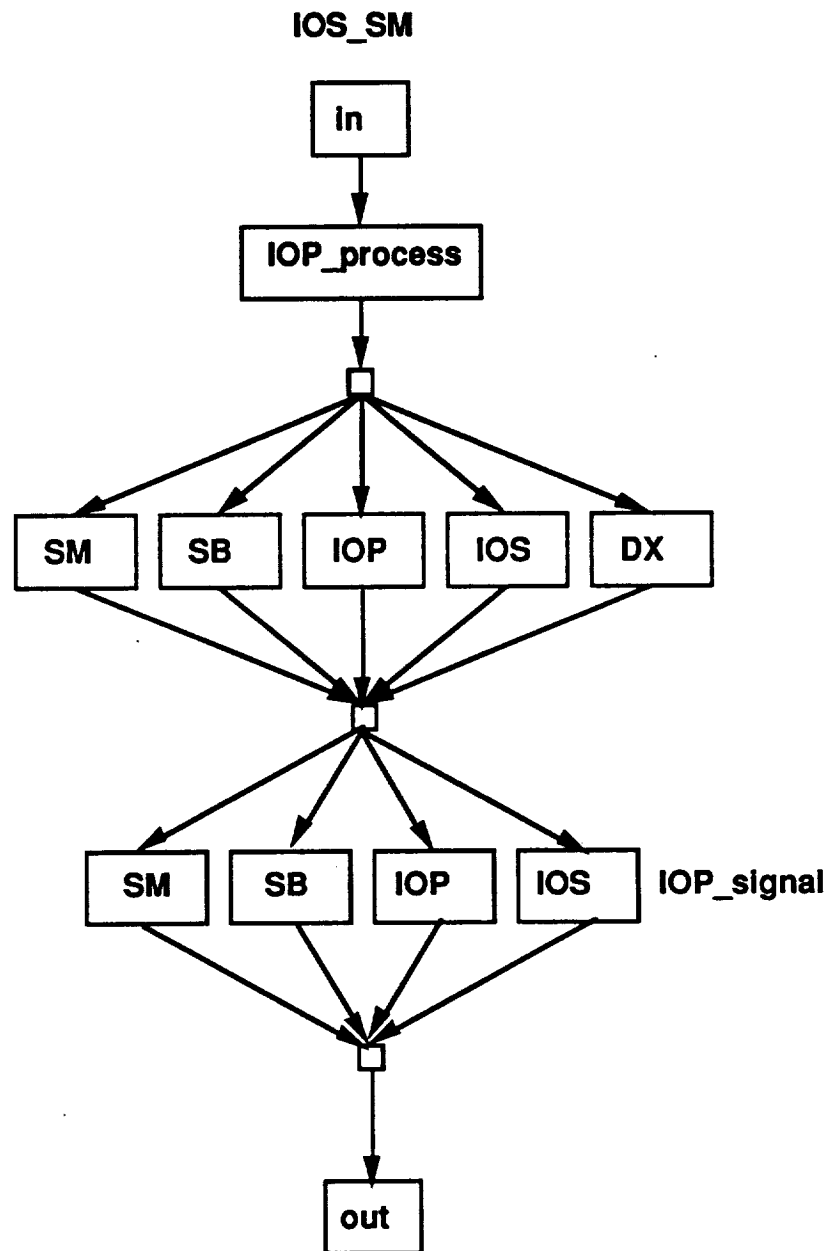


Figure 4-19. IOP Subgraphs (2)

The “net” subgraph models the process of executing the Input and Output process for the given frame. The “in” node in this subgraph receives a token from the higher-level scheduler graph; this token indicates the type of the current frame and consequently which I/O chain to execute. Operation of this graph is similar to the “app_tsk” graph described above. The “IOS Chain Mgmt” node models the operation of the I/O Sequencer (IOS), which is responsible for iteratively emitting tokens to the “xmit_trans” and the “read_trans” nodes, depending on the sequence of I/O transactions to be executed for the current frame. The IOS Chain Mgmt node emits exactly one token per firing; this token goes to either an

xmit_trans, a read_trans, or the “EOC” node. Each xmit_trans or read_trans node corresponds to an I/O transaction which can be performed by the candidate architecture, and represents the sum of times required for the IOS to transmit the transaction to the I/O device, the recipient node to respond, and the IOS to process the response. Refer to [2] for a more detailed description of the I/O transaction process. Upon completion of the transaction an xmit_trans or read_trans node emits a token to an “any” join node, which in turn provides a token to the IOS Chain Mgmt node. If the IOS Chain Mgmt node determines that the chain is complete for the current frame, it emits a token to the EOC node, which models the IOS’ End Of Chain processing. Upon completion of this processing, a token is emitted to the “out” node, which signals that the chain is complete to the higher-level graph.

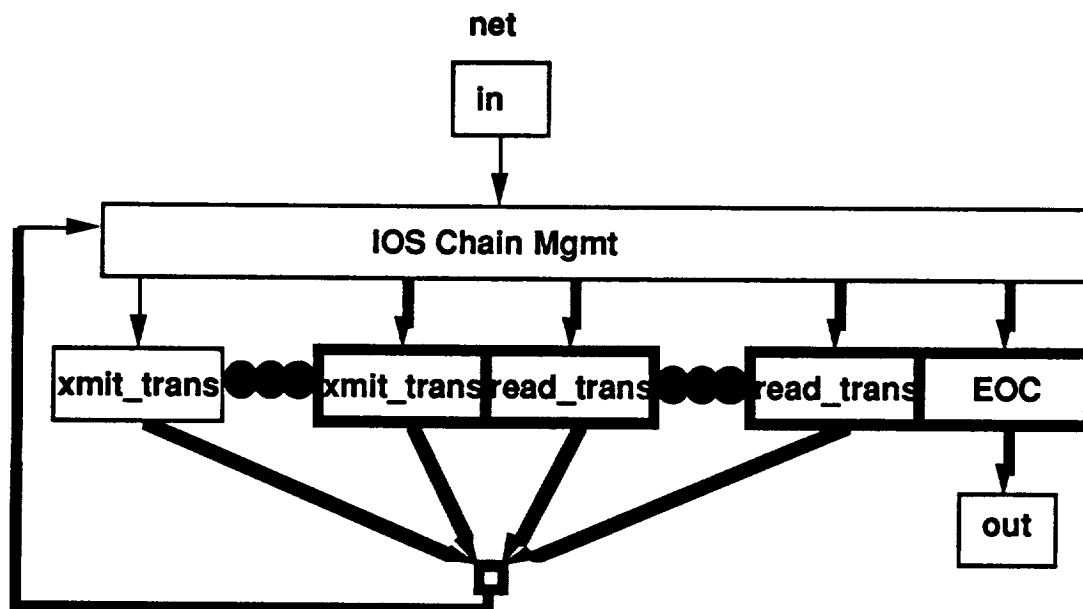


Figure 4-20. I/O Network Subgraph

4.3.4 Requirements and Performance Modeling Results

Based on the progress made in this phase, several conclusions and recommendations may be stated.

Fundamental numerical quantities needed for the detailed AIPS architectural synthesis and performance evaluation have been identified. For each task, the following parameters must be provided or otherwise generated: frame rate, throughput (or instructions per execution), throughput margin, processing lag, scheduling requirements (e.g., preemptible or nonpreemptible), task execution order dependencies, inter-function communication requirements (bits per iteration, latency), code/data memory, code/data memory margins, amount (number of bits) of input data required to each task for each frame iteration, number of bits, source (I/O device or producer task) of each input datum, maximum allowable latency and jitter for each input datum, amount (number of bits) of output data produced by each task for each frame iteration, destination(s) (I/O device or

consumer task) of each output datum, and maximum allowable latency and jitter for each output datum.

Preliminary ALS requirements were obtained from Martin Marietta Denver Aerospace. A technique was developed to process their requirements to generate the requisite parameters listed above. Partial conversion of the Martin Marietta requirements into these parameters was completed. Close interaction between CSDL and Martin Marietta during the requirements acquisition phase in our opinion enhanced obtaining an accurate understanding of the avionics system's computational requirements, and should be continued.

A two-level hierarchical ADAS model was constructed of a single FTP executing frame-based iterative control system such as that envisioned for the ALS avionics. This model is available for execution on a Sun workstation. However, the incompleteness of the ALS requirements received to date did not permit its use for modelling the ALS functions.

Complete requirement acquisition and use of the ADAS models will yield a quantitative determination of performance parameters such as input and output latency, processing lag, transport lag, control loop jitter, component utilization, and queue sizes.

4.3.5 Requirements and Performance Modeling Recommendations

While much progress was made in extracting ALS requirements from Martin Marietta, additional work needs to be done to obtain avionics requirements at the task-level granularity. Also, a mutually meaningful definition of I/O and Interfunction communication requirements needs to be determined and the relevant parameters quantified.

In the modeling effort, additional lower-level ADAS models of the detailed workings of the FTP, IOS, and I/O network need to be constructed. The two-level model presented herein need to be augmented by additional hierarchical levels. The model then needs to be expanded to model multiple FTPs communicating over the InterComputer network.

The task-granularity requirements and the multi-FTP ADAS models can then be used for estimation of the AIPS' performance. Before this can be done, however, allocation of the several ALS application functions to multiple FTPs must be performed. This process is currently manual, and replete with tedium and trial-and-error. Consequently, automated means of mapping the set of distributed functions to the FTPs should be investigated. Key to this effort would be the development of an automated interface between the mapping function and the ADAS evaluation tool.

5.0 EMPIRICAL TEST AND EVALUATION

5.1 Introduction

An important part of the AIPS knowledgebase is the performability characterization of the AIPS architecture. For real time aerospace applications, not only the computer performance but also the ability of the computer to deliver that performance in the presence of failures is of paramount importance. Performability, then, refers to the range of expected performance levels and their associated reliabilities or expectation probabilities.

The AIPS performability knowledgebase consists of analytical and empirical relationships between three major domains: performance, reliability and architectural parameters. There are many different figures of merit that can be used to measure the performance and reliability of an architecture or to compare the performability of several alternative architectures. The choice of which figures of merit to use depends on what is important in the intended application of the architecture. A set of metrics that are considered relevant for the broad range of mission- and safety-critical real time aerospace applications, whose requirements AIPS is designed to fulfill, are described in Section 5.3. A brief overview of the AIPS architecture and a set of parameters that characterize AIPS are contained in Section 5.2.

The performability knowledgebase, when it is completed, can be used by a system designer to configure the AIPS building blocks to meet a specific set of mission objectives and requirements. It can then also be used to predict the system behavior (in terms of performance, reliability, availability and other metrics) for a given hardware and software implementation without actually building and testing a prototype. This should greatly expedite the design and validation of mission- and safety-critical real time avionics systems.

Sections 5.4, 5.5 and 5.6 give an overview of three instantiations of the AIPS building blocks - a Fault Tolerant Processor, a fault-and damage-tolerant Input/Output Network, and a fault and damage tolerant inter-computer network - and their performability data that has been obtained to date. These include the overheads of a real time Ada® operating system, the redundancy management software and the interprocessor communication software. Throughput benchmarks as well as certain reliability related parameters such as fault detection and recovery time have been measured. Section 5.7 presents the impact of advanced software technology insertion. Finally, Section 5.8 concludes with a summary and status of the present work and future plans.

® Ada is a registered trade-mark of the Ada Joint Program Office.

5.2 AIPS Architectural Parameters

The goal of the Advanced Information Processing System is to develop an objective knowledgebase for achieving validatable fault tolerant distributed architecture designs applicable to a broad range of real time aerospace applications. Performance, reliability and operational requirements for seven such applications ranging from commercial transport aircraft to manned space platforms were surveyed and are summarized in [16]. An architecture was synthesized to not only meet the specific quantitative requirements of these applications but also possess a number of desirable qualitative attributes. These include transparency of fault tolerance to software, low overheads for redundancy management, expandability, graceful degradation and validity.

AIPS is a multicomputer architecture composed of hardware and software building blocks that can be configured to meet a broad range of application requirements. The hardware building blocks are fault tolerant, general purpose computers, fault- and damage-tolerant inter-computer (IC) and Input/Output (I/O) networks, and interfaces between the networks and the general purpose computers. Figure 5-1 illustrates the AIPS hardware building blocks. The general purpose computers designed for AIPS are termed Fault Tolerant Processors (FTPs) and may be simplex, duplex, triplex, or quadruplex units based on the reliability requirement of the application program being performed at the processing site. The software building blocks are the major system services: local system services, Input/Output system services, inter-computer communication services and the system manager. This software provides the services necessary in a traditional real time computer such as task scheduling and dispatching, communication with sensors and actuators, etc. The software also supplies the redundancy management services necessary in a redundant computer and the services necessary in a distributed system such as inter-function communication across processing sites, management of distributed redundancy, management of networks, and migration of functions between processing sites. The AIPS building blocks can be configured to provide an extremely wide range of performance and reliability as discussed earlier in Section 2.

The system designer can match these to his/her specific application requirements by choosing appropriate values for AIPS architectural parameters. For example, at the system level, the number of processing sites, the FTP redundancy level across processing sites, and the redundancy level of the IC Network are some of the parameters that are under the system designer's control. Other parameters are a function of the hardware and software technology insertion. Hardware technology insertion includes the raw CPU throughput, mean time between failures (MTBF) of CPU, memory and other hardware components, and raw bandwidth of I/O and IC networks. The software technology controls certain

basic parameters such as the Ada Rendezvous time which depends among other things on the Ada compiler technology.

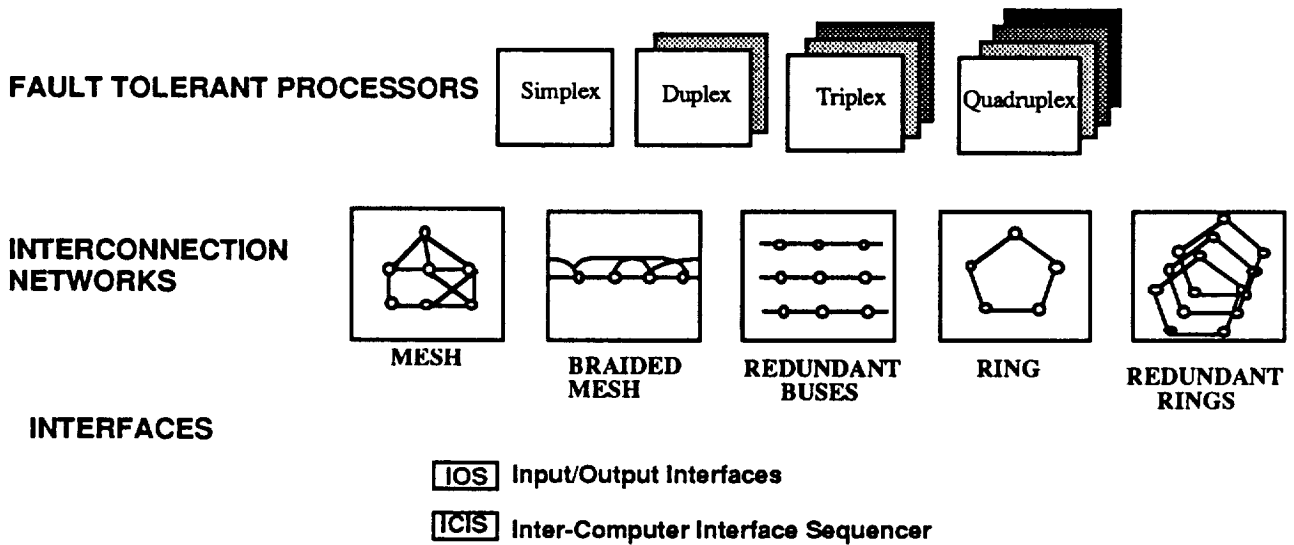


Figure 5-1. AIPS Hardware Building Blocks

Table 5-1 summarizes some of the salient AIPS architectural parameters that influence its performance and reliability. These parameters are divided along the AIPS building blocks into four major sections: 1. System, 2. FTP-Local System Services, 3. I/O Network - I/O System Services and 4. IC Network - IC Communication Services.

Table 5-1. AIPS ARCHITECTURAL PARAMETERS

1. System
 - 1.1 Number of Processing Sites
 - 1.2 FTP Redundancy Level Across Processing Sites
 - 1.2.1 All Triplex Only
 - 1.2.2 Simplex, Duplex, Triplex
 - 1.2.3 Simplex and Duplex Only
 - 1.3 IC Network Redundancy Level
 - 1.3.1 Simplex
 - 1.3.2 Duplex
 - 1.3.3 Triplex
 - 1.4 Homogeneity Across Processing Sites
 - 1.4.1 Identical FTPs

- 1.4.2 Diverse FTPs
 - 1.4.2.1 Heterogeneous ISAs
 - 1.4.2.2 Heterogeneous System Service Implementation
- 2. FTP-Local System Services
 - 2.1 FTP Redundancy Level
 - 2.2 Processors Per Channel
 - 2.3 Raw CPU Throughput
 - 2.3.1 Clock Frequency
 - 2.3.2 Data/Instruction Width
 - 2.3.3 Memory Speed
 - 2.3.4 Cache
 - 2.3.5 MMU
 - 2.3.6 FPU
 - 2.4 Hard Failure Recovery Time Distribution (Failure Detection, Isolation and Reconfiguration Distributions)
 - 2.4.1 Fast FDIR Frequency & Coverage
 - 2.4.2 Self Test Frequency & Coverage
 - 2.4.3 Memory Scrub Frequency & Coverage
 - 2.4.4 Watchdog Timer Coverage
 - 2.4.5 Monitor-Interlock Coverage
 - 2.5 Transient Failure Recovery Time Distribution (FDIR Distributions)
 - 2.5.1 Transient FDIR Frequency & Coverage
 - 2.5.2 Self Test Frequency & Coverage
 - 2.5.3 Memory Scrub Frequency & Coverage
 - 2.5.4 Lost Soul Synchronization Time
 - 2.6 Common Mode Failure (CMF) Recovery Distribution (FDIR Distributions)
 - 2.6.1 Hardware Common Mode Failures
 - 2.6.2 Redundancy Management (RM) CMF
 - 2.6.3 Non-RM CMF
 - 2.7 Redundancy Management (RM)
 - 2.7.1 Fast FDIR Frequency
 - 2.7.2 Fast FDIR Execution Time
 - 2.7.3 Transient FDIR Frequency
 - 2.7.4 Transient FDIR Execution Time
 - 2.7.5 Self Test Frequency
 - 2.7.6 Self Test Execution
 - 2.7.7 Memory Scrub Frequency
 - 2.7.8 Memory Scrub Execution Time
 - 2.7.9 Common Mode Failure Recovery Frequency

- 2.7.10 CMF Recovery Execution Time
- 2.8 Byzantine Resilience Overheads
 - 2.8.1 Data Exchange Time
 - 2.8.1.1 Fault Tolerant Clock Frequency
 - 2.8.1.2 Data Exchange Width
 - 2.8.2 No. of Simplex Source Words (Inputs) From Application Reports
 - 2.8.3 Input Frequency
- 2.9 Mean Time Between Failures (MTBFs)
 - 2.9.1 FTP MTBF
 - 2.9.1.1 CPU
 - 2.9.1.2 Memory
 - 2.9.1.3 Shared Bus Controller
 - 2.9.1.4 IOS
 - 2.9.1.5 ICIS
 - 2.9.1.6 Monitor-Interlock
 - 2.9.2 Ratio of Hard Failures to Transient Failures
- 3. IO Network-IO System Services
 - 3.1 Redundancy Management
 - 3.1.1 Network FDIR Time Distribution
 - 3.1.2 Initial Growth Time
 - 3.1.3 Worst-Case In-Flight Reconfiguration Time
 - 3.2 RM Overheads
 - 3.2.1 Number of Nodes
 - 3.2.2 Node Transaction Time
 - 3.2.3 Frequency of Node Polling
 - 3.2.4 Spare Link Test Time
 - 3.2.5 Number of Spare Links
 - 3.2.6 Frequency of Spare Link Testing
 - 3.2.7 Number of Ports Per Node
 - 3.2.8 Network Topology
 - 3.2.9 IOS Self Tests
 - 3.3 MTBFs
 - 3.3.1 Node
 - 3.3.2 Port
 - 3.3.3 Link
 - 3.4 Network Performance
 - 3.4.1 Contention Time
 - 3.4.2 Node Response Time
 - 3.4.3 DIU Response Time
 - 3.4.4 Bus Bandwidth

- 3.4.5 Transaction Completion Time
- 3.4.6 Chain Completion Time
- 3.4.7 I/O Request Completion Time
- 3.4.8 Network Physical Link Layer Protocol
- 3.4.9 Network Data Link Layer Protocol
- 4. Inter-Computer Network-Inter-Computer Communication Services
The inter-computer details are analogous to the IO.

5.3 Performance and Reliability Metrics

The choice of which figures of merit to use to measure the performance and reliability of an architecture or to compare the performability of several alternative architectures depends on what is considered important in the intended application of the architecture. As stated earlier, the performance, reliability and operational requirements for seven intended applications of AIPS were surveyed and analyzed [16]. A set of metrics have been chosen that are considered relevant for the broad range of mission and safety-critical real time requirements represented by these seven applications.

As with the architectural parameters discussed in the previous action, the metrics are partitioned and organized along the AIPS building block lines. Tables 5-2 and 5-3 summarize the salient performance and reliability metrics, respectively. At the system level, maximum useful throughput, communication latency between functions executing in different processing sites, and total useful I/O bandwidth are some of the important performance metrics. Throughput and memory resources devoted to various system services are also important since they measure the efficiency of the architecture. At each processing site, in addition to useful throughput and various overheads, there are two important metrics that measure the real time and fault tolerance related performance. From the real time viewpoint, the jitter in starting time of successive iterations of periodic tasks, the delay in starting aperiodic tasks and the transport lag between reading sensors and sending the corresponding actuator commands are important. From a fault tolerance viewpoint, the time to distribute data from one processor to other redundant processors in the FTP and to vote the outputs of redundant processors are important metrics. These last two metrics are not specific to the FTP but are equally applicable to any fault tolerant computer that uses redundant processors. As a matter of fact, all the metrics discussed here are quite general and apply to any distributed real time fault tolerant architecture.

Some of the metrics depend not only on the architecture but also on the application workload. For example, in order to compute the overheads of the Ada Run Time System and task scheduling, it is necessary to know the number of tasks to be scheduled in each frame, their iteration frequency and so on. The performance metrics for the AIPS FTP, the I/O Network, and the IC Network discussed in the next three sections have been evaluated

in the context of a specific workload that is considered representative of real time flight control applications.

The performance metrics for the input/output and the inter-computer communication are also listed in Table 5-2. In the context of reliability, the figures of merit of interest at the system level are mission success probability, mean time to repair, and maximum function reliability and availability. Some representative measures of merit, partitioned along the three AIPS building blocks, are also listed in Table 5-3.

Table 5-2. Performance Metrics

1. System
 - 1.1 Max Useful Throughput
 - 1.2 Throughput as a function of # Processing Sites
 - 1.3 Inter-function communication delay (non-colocated)
 - 1.4 Function Migration Latency
 - 1.5 Total I/O Bandwidth
 - 1.6 System FDIR Overheads (Throughput, Memory)
 - 1.7 Non-FDIR System Services Overheads (Throughput, Memory)
2. FTP-Local System Services
 - 2.1 Useful Throughput
 - 2.1.1 CP
 - 2.1.2 IOP
 - 2.2 Overheads (CP Throughput & Memory, IOP Throughput & Memory)
 - 2.2.1 Ada Run Time System
 - 2.2.2 Scheduler
 - 2.2.3 Redundancy Management
 - 2.2.4 Time Manager
 - 2.3 IOP-CP Communication
 - 2.3.1 Shared Memory Time
 - 2.3.2 Interrupt Time
 - 2.4 Data Exchange Time
 - 2.4.1 Simplex Source Broadcast
 - 2.4.1.1 Dedicated
 - 2.4.1.2 Shared
 - 2.4.2 Voted Exchange
 - 2.4.2.1 Dedicated
 - 2.4.2.2 Shared
 - 2.5 Periodic Tasks
 - 2.5.1 Jitter
 - 2.5.2 Transport Lag

- 2.6 Aperiodic Tasks
 - 2.6.1 Start Delay
- 3. I/O System Services
 - 3.1 Memory Mapped I/O
 - 3.1.1 Sensor Read Latency
 - 3.1.1.1 Private Bus
 - 3.1.1.2 Shared Bus
 - 3.2 Network I/O
 - 3.2.1 Dedicated Network
 - 3.2.1.1 Sensor Read Time
 - 3.2.1.1.1 IOP Read Time
 - 3.2.1.1.2 IOP-CP Latency
 - 3.2.1.2 I/O Redundancy Management Overhead
 - 3.2.1.2.1 Network Manager
 - 3.2.1.2.2 IOS Self Tests
 - 3.2.1.2.3 Maintenance & Repair Routines
 - 3.2.1.3 Max Useful Bandwidth
 - 3.2.2 Shared Network
 - 3.2.2.1 Sensor Read Time
 - 3.2.2.2 I/O Redundancy Management Overhead
 - 3.2.2.3 Max Useful Bandwidth
 - 3.2.3 Max # Nodes/Network
 - 3.2.4 # DIUS/Node
- 4. Inter-Computer Communication Services
 - 4.1 Useful Bandwidth
 - 4.2 Inter-Function Communication
 - 4.2.1 Broadcast (one to all) Latency
 - 4.2.2 Multicast (one to many) Latency
 - 4.2.3 One-to-one Latency
 - 4.2.3.1 Mailbox
 - 4.2.3.2 With Acknowledgement
 - 4.2.4 Overheads
 - 4.3 System Time Manager
 - 4.3.1 Synchronization Granularity
 - 4.3.2 Synchronization Accuracy
 - 4.3.3 Overheads
 - 4.4 Max # Nodes Per Layer
 - 4.5 Byzantine Resiliency Overheads
 - 4.5.1 All Triplex Configuration
 - 4.5.1.1 Reduce Input Data to Congruent Value

- 4.5.1.2 Failure Detection & Isolation
 - 4.5.2 Mixed Redundancy Configuration
 - 4.5.2.1 Reduce Input Data to Congruent Value
 - 4.5.2.2 Failure Detection & Isolation
- 4.6 Network Management Overheads
 - 4.6.1 IC Network Manager
 - 4.6.2 ICIS Self Tests
 - 4.6.3 Network Maintenance & Repair
- 4.7 ISO Layer Latency & Overheads
 - 4.7.1 Physical Link Layer
 - 4.7.2 Data Link Layer
 - 4.7.3 Network Layer
 - 4.7.4 Transport Layer

Table 5-3. Reliability Metrics

1. System
 - 1.1 Max. Function Reliability (time)
 - 1.2 Max. Function Availability
 - 1.3 Mission Success Probability
 - 1.4 Mean Time To Repair (MTTR)
2. FTP-Local System Services
 - 2.1 FTP Failure Probability (time)
 - 2.1.1 Catastrophic Failure
 - 2.1.2 Degraded Mode
 - 2.1.3 Connectivity to IC Network (Failure Prob.)
 - 2.1.4 Connectivity to I/O Network(s) (Failure Prob.)
 - 2.2 FTP MTTR
3. I/O System Services
 - 3.1 Prob. of Accessing DIU
 - 3.2 Prob. of Accessing Network
 - 3.3 Prob. of Accessing Redundant Networks
 - 3.4 Network Outage
 - 3.4.1 Prob. of Outage
 - 3.4.2 Prob. Distribution of Outage Time
4. Inter-Computer Communication Services
 - 4.1 Prob. of N Critical Functions Communicating
 - 4.1.1 Fault Masking Communication
 - 4.1.2 Degraded Mode Communication (Not Fault Masking)
 - 4.1.3 Loss of Communication
 - 4.2 Prob. of a Critical Function and a Non-Critical Function Communicating
 - 4.3 Prob. of Two Non-Critical Functions Communicating
 - 4.4 Inter-Computer Network Outage
 - 4.4.1 Prob. of Outage
 - 4.4.2 Max. Time of Outage or Prob. Distribution of Outage Time

5.4 FTP Empirical Knowledgebase

5.4.1 FTP Architecture and H/W and S/W Implementation Technology

The Fault-Tolerant Processor (FTP) consists of a variable number of redundant processing channels depending on the reliability requirements of the application. The AIPS engineering model FTP is intended to be operated primarily as a triplex, but a single channel can also be used for non-critical operations as a simplex computer.

Each channel of an FTP consists of three sections: a computational section, an input/output section, and the resources shared between them. The first section contains a Computational Processor (CP), memory, and interval timers. The second section contains an Input/Output Processor (IOP), memory, and interval timers. The shared resources include shared memory, data exchange hardware, a system timer, a monitor interlock, a fault tolerant clock, and external interface hardware. The redundant processors are tightly synchronized using the fault-tolerant clock. Data is exchanged among redundant channels on point-to-point links. The data exchange hardware also performs the bit-for-bit voting, fault detection and masking functions in a manner that satisfies all the requirements to protect the FTP from Byzantine failures [33].

A functional view of one channel of an AIPS FTP is shown in Figure 5-2. The CP and IOP are identical, conventional processor architectures. Interval timers are used for scheduling tasks and maintaining time-out limits on applications tasks (task watchdog timers). A hardware watchdog timer is provided to increase fault coverage and to cause a processor to fail-safe in case of hardware or software malfunctions. This timer resets the processor and disables all of its outputs, if it is not reset periodically. The watchdog timer is implemented independently of the basic processor timing circuitry. A monitor and interlock circuit in each channel provides the capability to disable the outputs of faulty processors. Any two correctly operating processors in a triplex FTP can disable the outputs of the third failed processor through this interlock mechanism. A processor that is failed active is thus prevented from transmitting erroneous data or commands on I/O networks, IC networks, and local I/O devices.

The CP and IOP share resources through a bus that can be accessed by either processor. These shared resources include memory; a system timer; the fault tolerant clock; the monitor interlock, the interchannel data exchange and voting circuits; interfaces to one or more I/O networks; and interfaces to the IC network. The fault tolerant clock monitors and adjusts the individual SYSCLKs of the redundant processors so that they remain synchronized at the instruction level. The system timer is derived from the fault tolerant clock and will serve as a real-time clock.

All of the AIPS system software has been written in Ada. The same software executes on a redundant FTP as on a simplex channel and application code is written as if it were to operate on a simplex computer. All redundant processors have identical software and execute identical instructions at exactly the same time. The IOP and CP communicate through the shared memory. The IOP and CP have independent operating systems that cooperate to assure that the data from input devices is made available to the applications programs running in the CP in a timely and orderly fashion. Similarly, the two processors cooperate on the outgoing information so that the output devices receive commands at appropriate times. Hence the CP and IOP actions must be synchronized to some extent. To help achieve this synchronization in software, a hardware feature has been provided which enables one processor to interrupt the other. By writing to a reserved address in shared memory the CP can interrupt the IOP and by writing to another reserved location the IOP can interrupt the CP. Different meanings are assigned to this interrupt by leaving an event code in some other predefined part of the shared memory, before the inter-processor interrupt is asserted.

For routine flow of information in both directions, the shared memory is used without interrupts but with suitable locking semaphores to pass consistent data sets. The interrupts can be used to synchronize this activity as well as to pass time critical data that must meet tight response time requirements. In order to assure data consistency, it is necessary that while one side is updating a block of data the other side does not access that block of data. This has been implemented using software semaphores. Hardware support for semaphores is provided in the form of the test and set instruction.

The architectural approach described above provides several significant operational benefits. The most important of these is the decoupling of the computational and input/output streams of transactions. The computational processor is unburdened from having to do I/O transactions. To the CP, all I/O appears memory mapped including not only I/O devices but also all other computers in the system. That is, each sensor, actuator, switch, computer, etc., with which the FTP interfaces can be addressed simply by using the I/O System Services calls to read and write words in shared memory.

The AIPS system software, as well as the hardware, has been designed to provide a virtual machine architecture that hides hardware redundancy, hardware faults, multiplicity of resources, and distributed system characteristics from the applications programmer. The particular module of the AIPS system software that is responsible for the FTP is called local system services [1]. The local system software is responsible for the operating system, interprocessor communication (CP, IOP) and the redundancy management of each FTP site. The local system services are: FTP initialization, real-time operating system, FTP Status Reporting, FTP Fault Detection, Isolation, and Reconfiguration (FDIR).

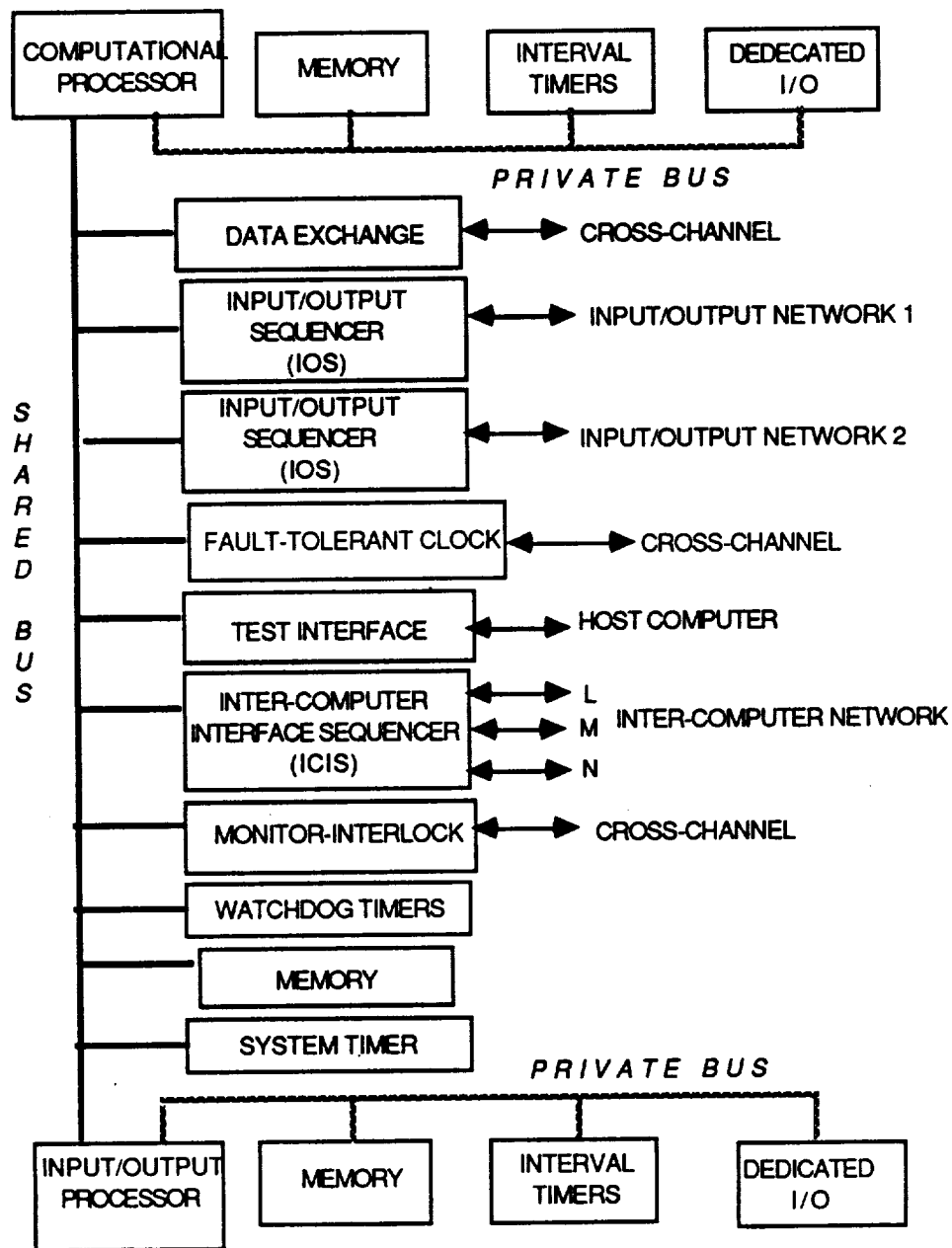


Figure 5-2. Fault Tolerant Processor Architecture: Functional View (One Channel)

The FTP real-time operating system supports task execution management, including scheduling according to priority, time and event occurrence, and is responsible for task dispatching, suspension and termination. It also supports memory management, software exception handling, and intertask communication between companion processors (IOP and CP). The FTP operating system resides on both the Computational Processor (CP) and the

Input/Output Processor (IOP). It uses the vendor-supplied Ada Run Time System (RTS), and includes additional features required for the CSDL FTP real-time bi-processor operating system. It is also responsible for providing time services to all users.

FTP FDIR has the responsibility for detecting and isolating hardware faults in the CPs, IOPs, and shared hardware. It is responsible for synchronizing both groups of processors in the redundant channels of the FTP and for masking and disabling outputs of failed channel(s) through mask registers and interlock hardware. After synchronization, all CPs will be executing the same machine language instruction within a bounded skew, and all IOPs will be executing the same machine language instruction within a bounded skew. FTP FDIR logs all faults and reports status to the FTP status reporter. It is responsible for the CPU hardware exception handling. It is also responsible for transient hardware fault detection and for running low priority self tests to detect latent faults. This redundancy management function is transparent to the application programmer.

5.4.2 FTP Performance Data

The performance of the AIPS Fault Tolerant Processor was measured empirically in order to characterize overheads of various system services [13]. Overhead data for Ada Run Time System (RTS) and the scheduler, referred to in the last section as the FTP real-time operating system, was gathered. Two implementations of the AIPS Fault Tolerant Processor (FTP) were used for the data collection. The first engineering model, referred to here as FTP Engineering Model 1, is a triplex FTP fabricated with 68010 microprocessors and a 7.8 MHz clock. The second engineering model, referred to here as FTP Engineering Model 2, is a quadruply redundant FTP with Motorola 68020 microprocessors, Motorola 68882 floating point co-processors, and a 14.5 MHz clock.

Figures 5-3 and 5-4 are diagrams of the Ada Run Time System (RTS) and scheduler overheads. The Ada RTS executing on both models was the RTS and scheduler software compiled with Verdix 5.4. Following are graphical representations of the task-switching overheads for FTP Model 1 (Figure 5-3) and FTP Model 2 (Figure 5-4). The rendezvous is a standard feature of the Ada language; the other three items are Draper enhancements to the Verdix supplied RTS which allows alternative ways of initiating tasks. A local event allows a user (task) to start a task on the same processor; a remote event allows a task to be started on the companion processor. A timer dispatch allows cyclic scheduling of a task. This is not possible with the standard Ada delay statement.

- Verdex 5.4 Run Time System
- 68010 processor @ 7.8 MHz.

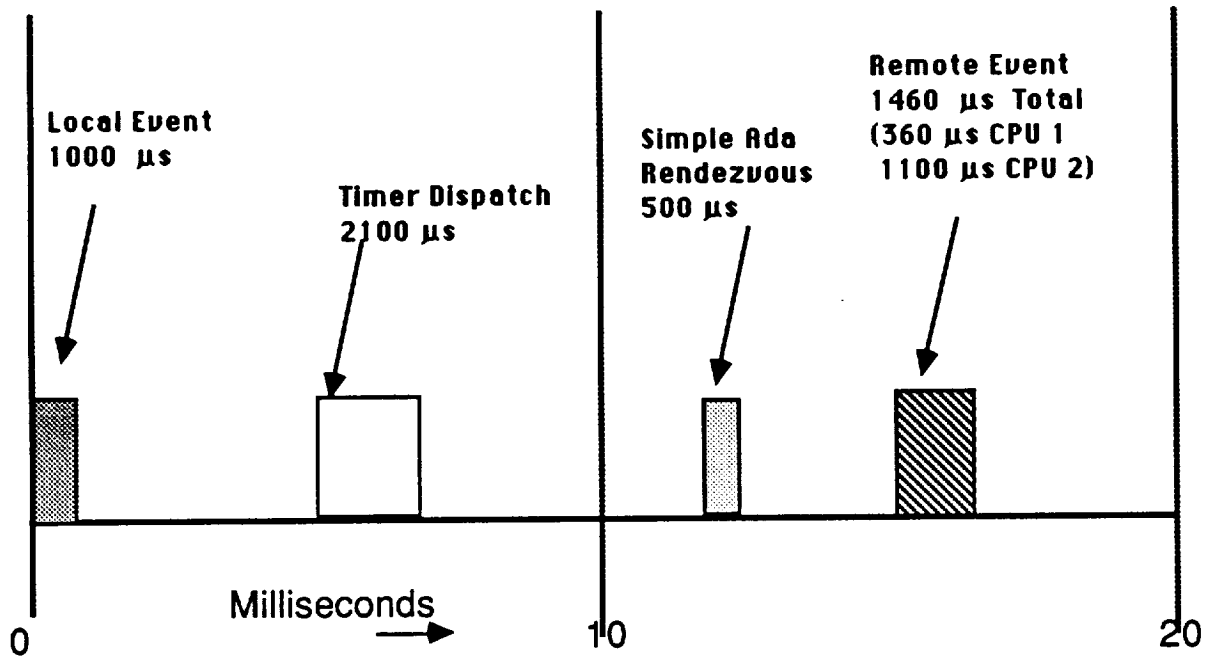


Figure 5-3. Ada Run Time System & Scheduler Overheads for FTP Model 1

The Ada RTS and scheduling overheads for FTP Model 1 are approximately 23% of a typical flight control frame of 40 millisecond with an average mix of 4 timer dispatches and 1 local event dispatch per frame. For FTP Model 2 the operating system overheads are approximately 6% of a 40 millisecond frame with an average mix of 4 timer dispatches and 1 local event dispatch per frame. With advanced technology insertions and a mature Ada compiler, operating system overheads will be an insignificant fraction of the processor throughput, as projected in Section 5.7.

5.4.3 FTP Reliability Parameters

Data was also collected in order to characterize the performance overheads of fault tolerance/redundancy management for the AIPS engineering models. Several of the parameters that characterize the reliability and that are used in the reliability models were measured empirically.

Two periodic FDIR tasks execute on each processor: Fast FDIR, which runs at 25 Hz and Transient FDIR, which runs at 4.16 Hz. Figure 5-5 is a graph of the redundancy management overhead under no fault conditions for FTP Model 1 and FTP Model 2. The overhead on Model 1 for Fast FDIR, which takes 2.4 milliseconds for each iteration on the CP and 2.0 for each iteration on the IOP, is 5-6% of the 40 millisecond frame under

nominal no-fault conditions. The horizontal lines represent time in milliseconds. The top horizontal line is the the time line for the CP and the bottom line is the time line for the IOP. The darker vertical lines represent the time period of the redundancy management tasks. Every 40 milliseconds Fast is executing on both the CP and IOP approximately 10 milliseconds out of phase with each other. Every 240 milliseconds both Fast and Transient execute. The time between 240 and 280 is blown up for clarity. The bottom half is a graph of the redundancy management overhead under no fault conditions for FTP Model 2.

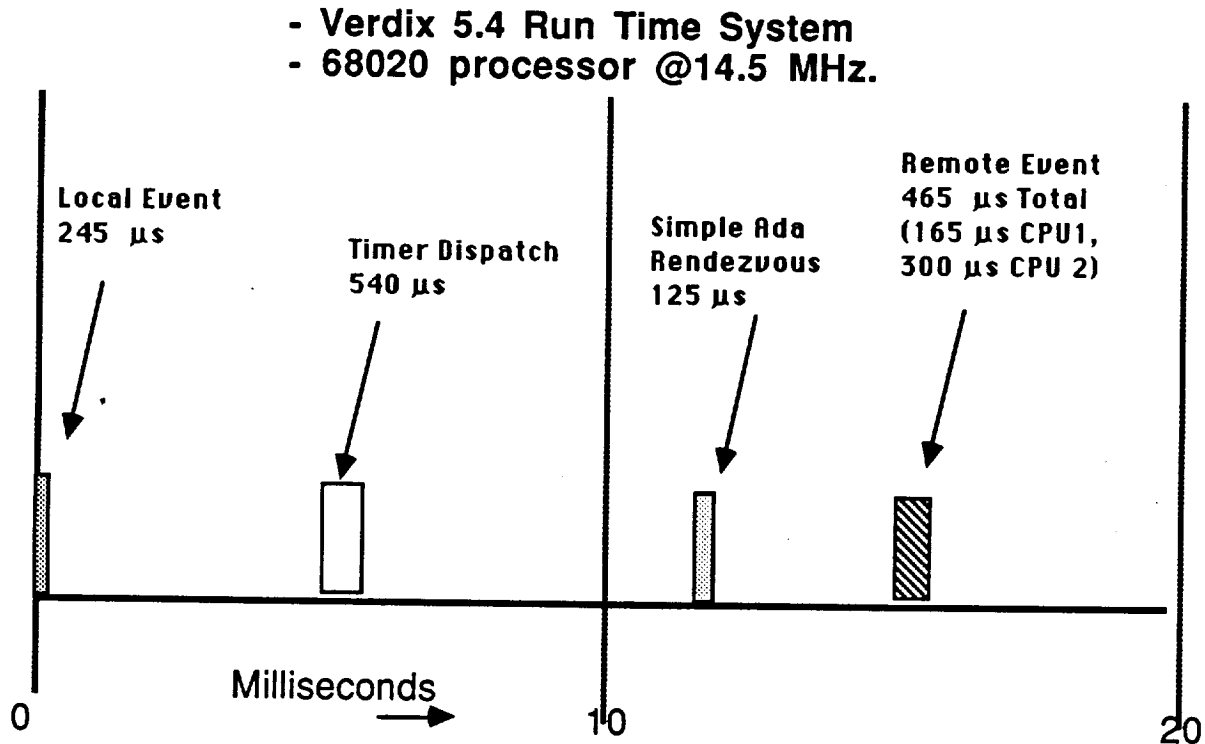


Figure 5-4. Ada Run Time System & Scheduler Overheads for FTP Model 2

The overheads for redundancy management with FTP Model 2 were reduced to only 2% of a 40 millisecond frame under nominal no-fault conditions.

Measurements were also taken under several faulty conditions. Figure 5-6 is a graph of the overhead for fault identification and recovery when a fault occurs in the data exchange network. A data exchange fault is detected through the analysis of error latches done by Fast FDIR. It does not desynchronize the faulty channel. In Figure 5-6 the time lines are divided into three stages: 1) when the fault is actually identified, 2) the time between fault identification and channel repair and 3) after channel has been repaired and recovered but it is in the trial period. In the fault identification stage, Fast FDIR takes 18%

of a 40 millisecond frame (up from 6% in the no-fault condition). During the time between fault identification and channel recovery, Transient FDIR increases to 2.1% (from less than 1% under no-fault conditions). Immediately after channel recovery, during the trial period, Transient FDIR increases to 8.6% (from less than 1% under no-fault conditions). The graph for FTP Model 2 shows these times reduced by a factor of 3 to 4.

Figure 5-7 is a graph of the overhead for fault identification and recovery when a fault occurs that causes a channel to desynchronize. Unlike the data exchange fault Fast FDIR on both the CP and the IOP detects the fault as shown in Figure 5-7. The times are comparable to those of the data exchange fault.

These measurements indicate the overhead parameters for fault detection, identification and recovery of a unsynchronized channel with FTP Model 2 were reduced to only 5% of a 40 millisecond frame. With further technology insertions and a mature Ada compiler redundancy management overheads will be an insignificant fraction of total processor throughput, as projected in Section 5.7.

5.5 Input/Output Network Empirical Knowledgebase

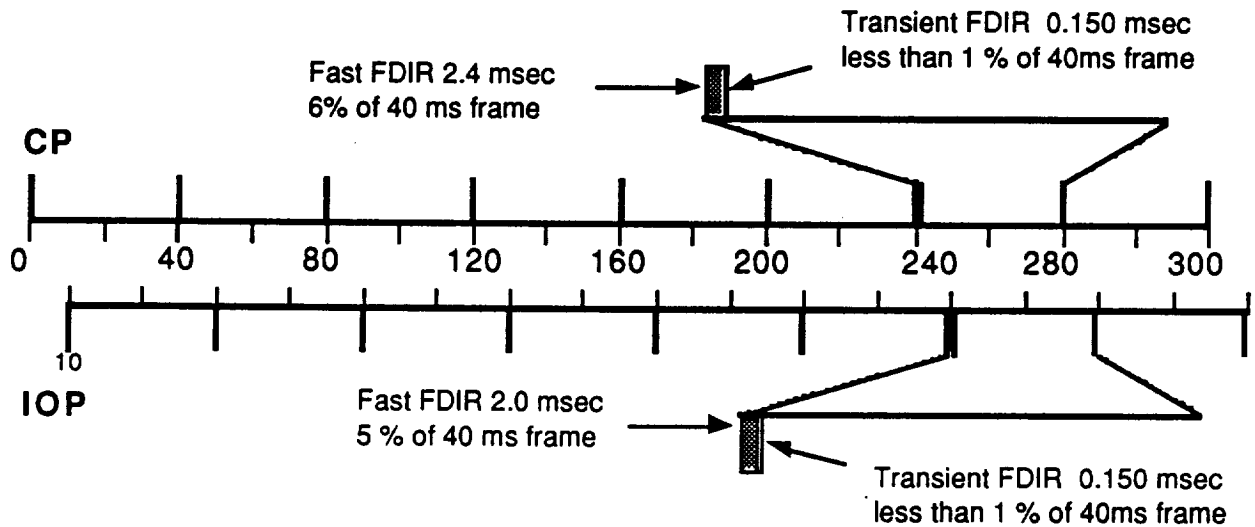
5.5.1 I/O Network Architecture and Hardware and Software Implementation Technology

For communication between a FTP and I/O devices, a damage and fault tolerant network is employed. The network consists of a number of full duplex links that are interconnected by circuit switched nodes. In steady state, the circuit switched nodes route information along a fixed communication path, or 'virtual bus', within the network, without the delays which are associated with packet switched networks. Once the virtual bus is set up within the network the protocols and operation of the network are similar to typical multiplex buses. Every transmission by any subscriber on a node is heard by all the subscribers on all the nodes, as though they were all linked together by a linear bus.

Although the network performs exactly as a bus, it is far more reliable and damage tolerant than a linear bus. A single fault or limited damage can disable only a small portion of the virtual bus, typically a node or a link connecting two nodes. The network is able to tolerate such faults due to the richness of interconnections between nodes. By reconfiguring the network around the faulty element, a new virtual bus is constructed. Except for such reconfigurations, the structure of the virtual bus remains static.

The nodes are sufficiently intelligent to recognize reconfiguration commands from the network manager, which is resident in one of the FTPs. The network manager performs the necessary diagnostics to identify the failed element and can change the bus topology by sending appropriate reconfiguration commands to the affected nodes.

**FTP Model 1
68010 7.8 MHz Triplex**



**FTP Model 2
68020 14.5 MHz Quad**

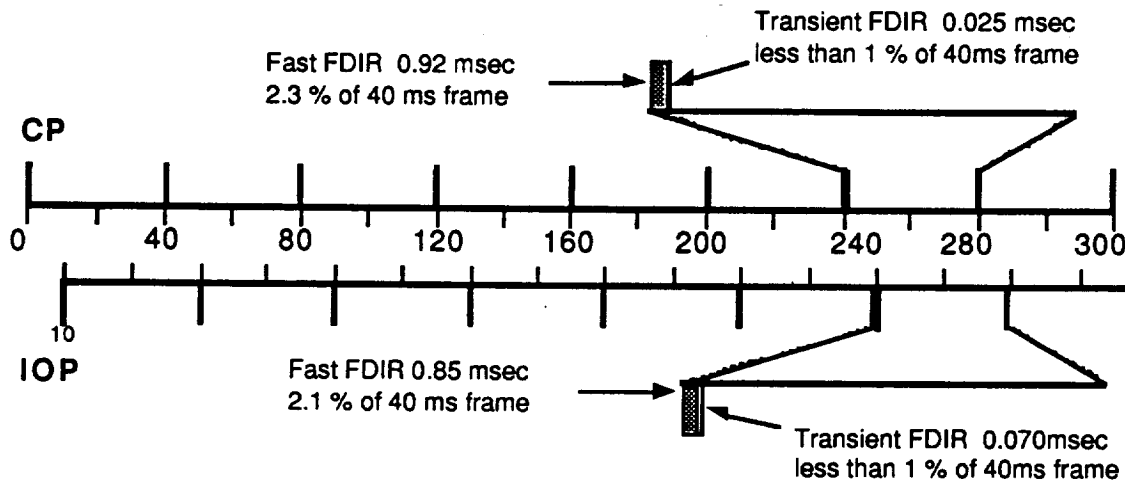
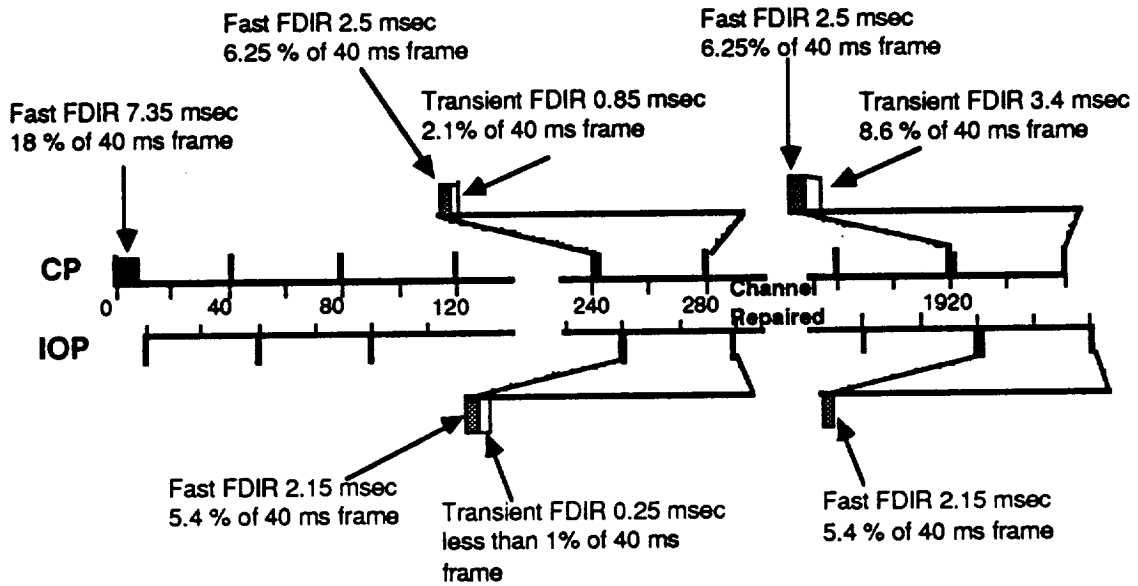


Figure 5-5. Redundancy Management Overhead - No Fault Conditions

FTP Model 1 68010 7.8 MHz Triplex



FTP Model 2 68020 14.5 MHz Quad

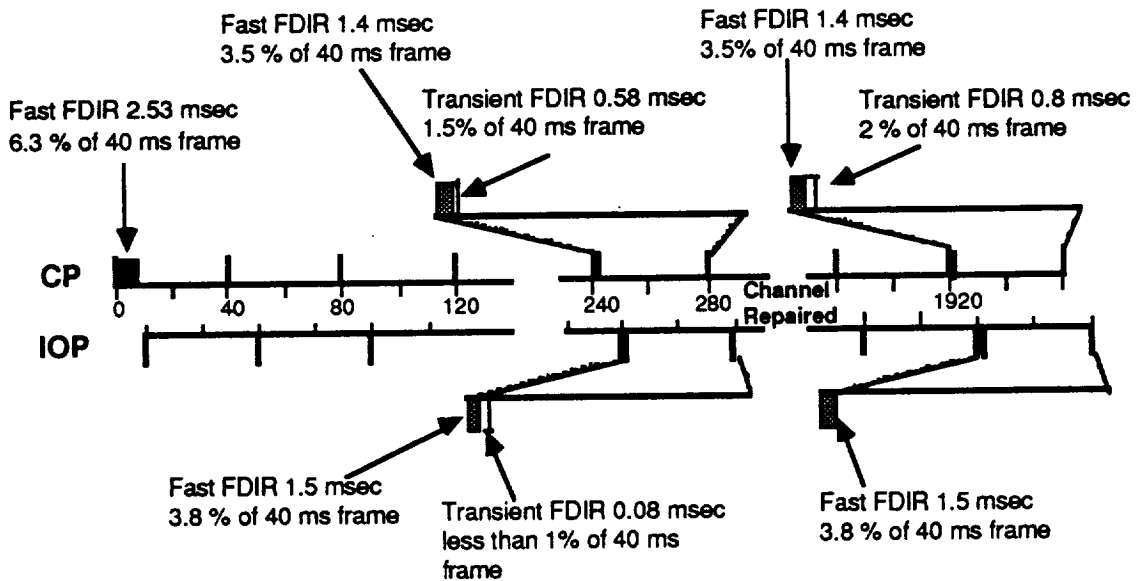
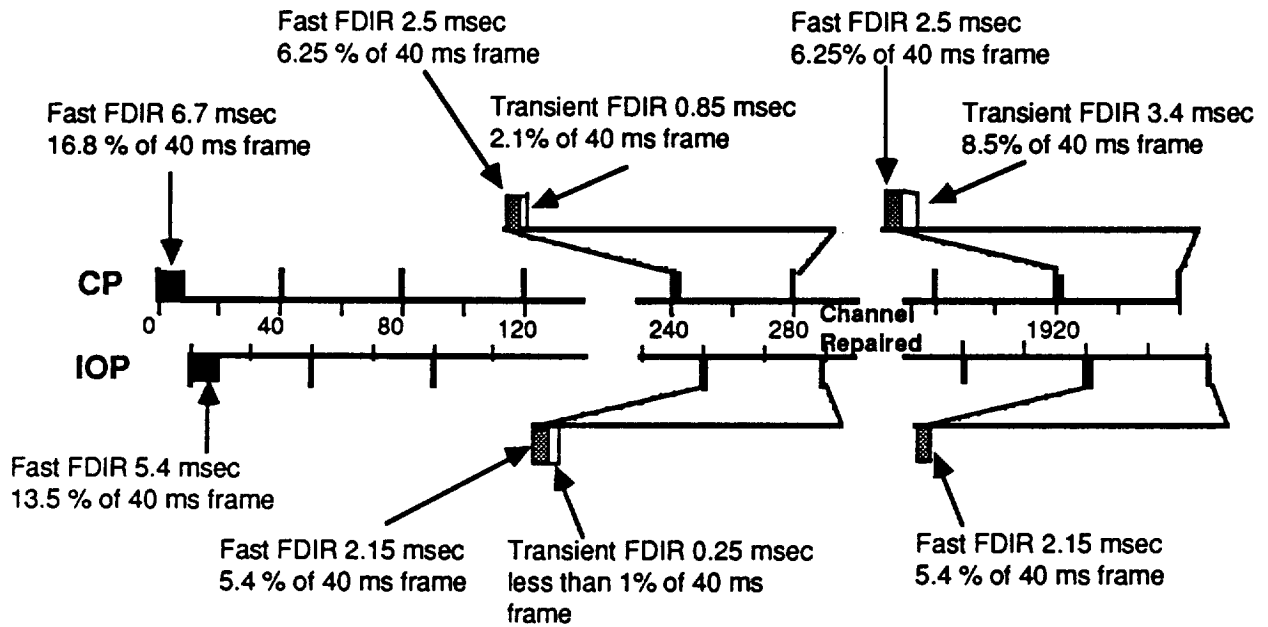


Figure 5-6. Redundancy Management Overhead - Data Exchange Fault

FTP Model 1 68010 7.8 MHz Triplex



FTP Model 2 68020 14.5 MHz Quad

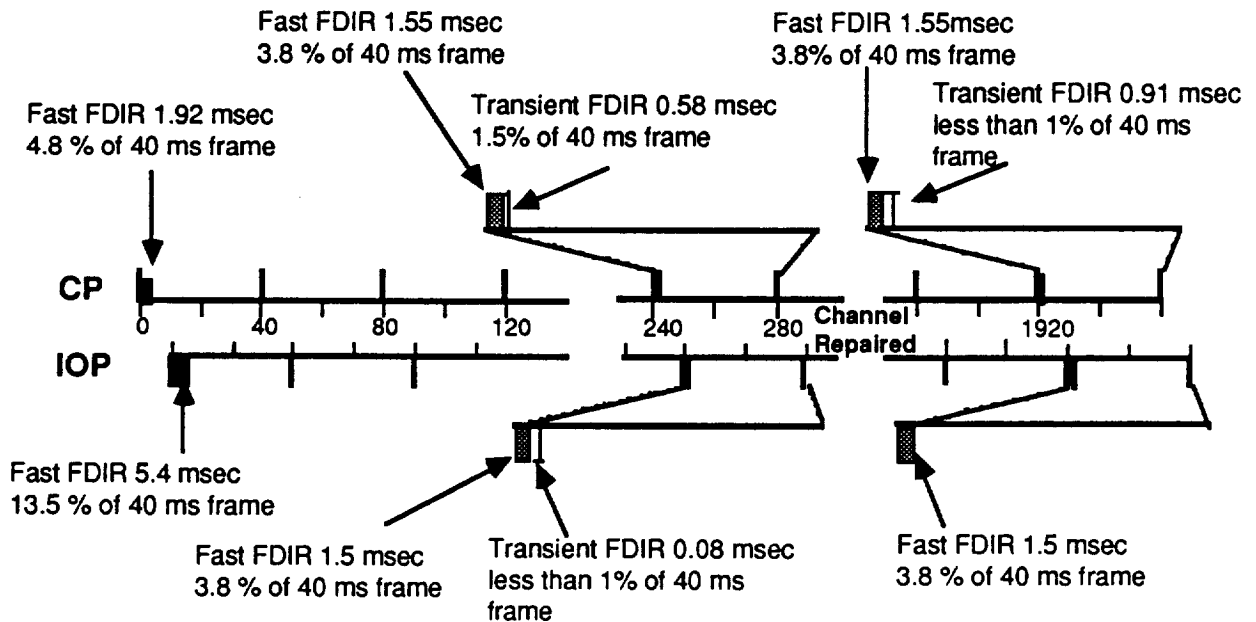


Figure 5-7. Redundancy Management Overhead - Unsynchronized Channel

Hardware and software for the AIPS architecture has been designed and implemented in two phases. The first phase completed was the centralized AIPS configuration. The centralized AIPS architecture, as shown in Figure 5-8, is configured as one triplex Fault Tolerant Processor (FTP), an Input/Output network and the interfaces between the FTP and the network, referred to as input/output sequencers (IOS). The laboratory demonstration of the input/output network consists of 15 circuit-switched nodes which can be configured as multiple local I/O networks connected to the triplex FTP. For example, the I/O network may be configured as one 15-node network, as shown in Figure 5-8, or as three 5-node networks.

The system software responsible for the I/O network is called Input/Output (I/O) System Services. The I/O system services provide efficient and reliable communication between the user and external devices (sensors and actuators). The I/O system services software is also responsible for the fault detection, isolation and reconfiguration of the I/O network hardware and FTP/network interface hardware (input/output sequencers). I/O system services is made up of three functional modules: I/O user interface, I/O communication management and the I/O network manager

The I/O user interface provides a user with read/write access to I/O devices or Device Interface Units (DIUs), such that the devices appear to be memory mapped. It also gives the user the ability to group I/O transactions into chains and I/O requests, and to schedule I/O requests either as periodic tasks or on-demand tasks. A detailed description of the I/O user interface is provided in [2].

The I/O communication manager provides the functions necessary to control the flow of data between a FTP and the various I/O networks used by the FTP. It also performs source congruency and error detection on inputs, voting on all outputs, and reports communication errors to the I/O Network Manager. It is also responsible for the management of the I/O request queues. A detailed description of the I/O communication manager is provided in [2].

The I/O Network Manager is responsible for detecting and isolating hardware faults in I/O nodes, links, and interfaces and for reconfiguring the network around any failed elements. The network manager function is transparent to all application users of the network. A detailed description of the I/O Network Manager is provided in [3].

5.5.2 I/O Network Performance Data

Performance measurements were done in order to determine the overheads for I/O communication management [13]. The data gathered to date was collected on FTP Model 1 using the Verdex 5.4 Ada compiler. The I/O communication manager is responsible for the bi-directional transfer of data from the application tasks executing on the CP via shared

memory to the I/O request queue on the IOP and then to the I/O devices on the I/O network. The application tasks are scheduled on the CP and the I/O requests are scheduled on the IOP.

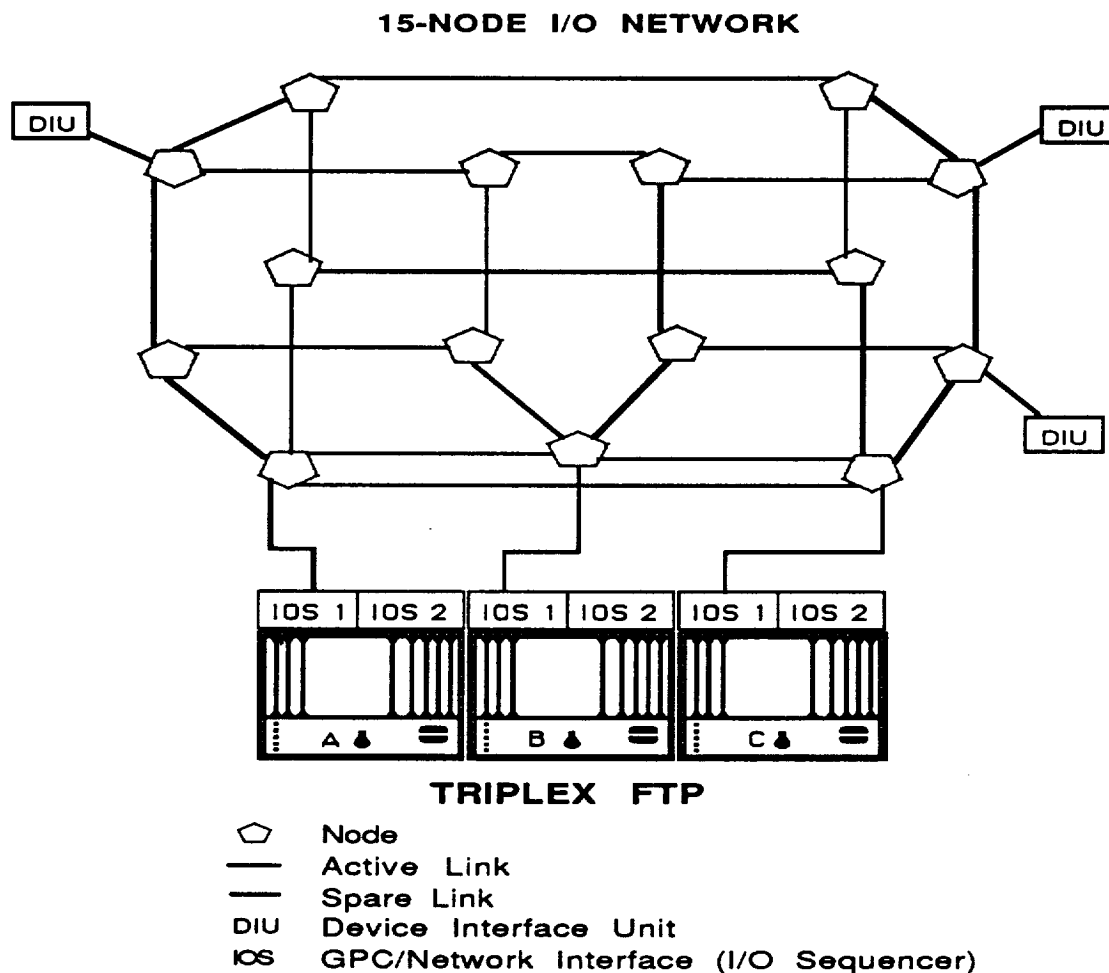


Figure 5-8. Centralized AIPS Configuration

The typical flight control application that was used for the performance data collection required three tasks: A periodic task scheduled to execute at 10 Hz, an on-demand task scheduled to execute at 5 Hz, and another on-demand scheduled to execute at 2.5 Hz. The periodic task started the on-demand tasks by using the local event provided by the operating system (see Section 5.4.2). Each of the three application tasks is synchronized with an on-demand I/O request. The synchronized application tasks and I/O requests are the following:

- a) The 10 Hz task starts a two chain I/O request which has 8 input transactions per chain.
- b) The 5 Hz task starts a two chain I/O request which has 10 input transactions per chain.

- c) The 2.5 Hz task starts a two chain I/O request which has 2 input transactions per chain.

The I/O request processing times for the three application tasks of a typical flight control application and FDIR are illustrated in Figure 5-9. The breakdown of the 10 Hz I/O request associated with the 10 Hz application task and executing as part of the I/O communication management on the IOP is illustrated in the timing diagram shown in Figures 5-10. The time on the network is indicated by the shaded area of this diagram. The performance data verifies that the dominating factors in the I/O communication overhead are the processor, compiler and the data exchange, not the network or the nodes. A typical flight control application on FTP Model 1 is able to execute at 10 Hz. The increased performance that will be gained using a state-of-the-art processor is discussed in Section 5.7.

5.5.3 I/O Network Reliability Parameters

Data was also collected in order to characterize the performance overheads of redundancy management for the AIPS I/O network. Several of the parameters that characterize the reliability were measured in order to use as actual input to our reliability models.

The network manager is responsible for the initial network growth, which at power-on involves a complete suite of network hardware testing. After initial growth of the I/O network, the network manager is responsible for the fault detection, isolation and reconfiguration of faulty nodes, links or I/O sequencers (IOS) and spare link cycling. Figure 5-11 indicates, for each type of network fault, the time period from the detection of a fault until the reconfiguration around the faulty component. The functions indicated by the outlined letters are illustrated in detail in Figures 5-12 and 5-13. The majority of network faults can be isolated with a simple node status collection chain. Several types of network faults combined with the network state cannot be isolated to the correct node, link or IOS with a status collection and analysis. Therefore, a network growth is necessary. Under the heading of network growth, three types of growth have been timed. The first growth with full diagnostics is only done at initial power-on. The second type of growth is done in order to isolate a babbling node or a node that fails active. The single chain growth is done to isolate under the following conditions:

- 1) A root link switch is required but zero spares are available.
- 2) A bad IOS corrupts the topology of the network such that the active network path is inconsistent with the network database.
- 3) No active root link exists after network reconfiguration.

- 4) An unexpected event occurs during network reconfiguration.
- 5) The network is not active and a channel comes back on-line.
- 6) A fault in the network occurs during the cycling of a spare element.
- 7) The cycling of a spare link detects that the link is failed. Accordingly, the network is returned to its state prior to the reconfiguration attempt, yet a fault occurs during the attempt to restore the previous state.
- 8) An unexpected event occurs during the restoration of a network element.

Figure 5-12 illustrates the overhead involved in the isolation and detection of a failed leaf (end) node, and Figure 5-13 illustrates the timing breakdown of the single chain grow. In each figure the total time is indicated by the top horizontal line.

As indicated by the diagrams, the time on the network does not cause extensive redundancy management performance penalties. Instead, the performance penalties are caused by the processor, compiler and data exchange hardware. A state-of-the-art processor and a mature Ada compiler will allow the network to recover from a fault in real time. The impact of technology insertion upon performance is discussed in Section 5.7.

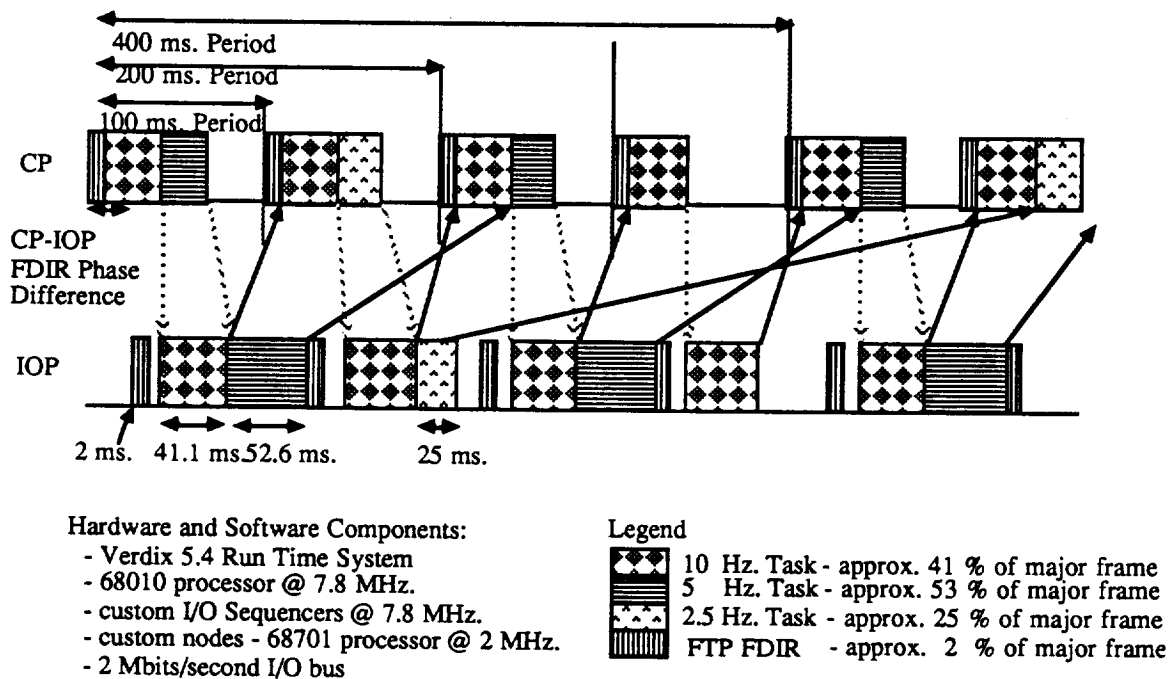


Figure 5-9. I/O Communications Management Overheads

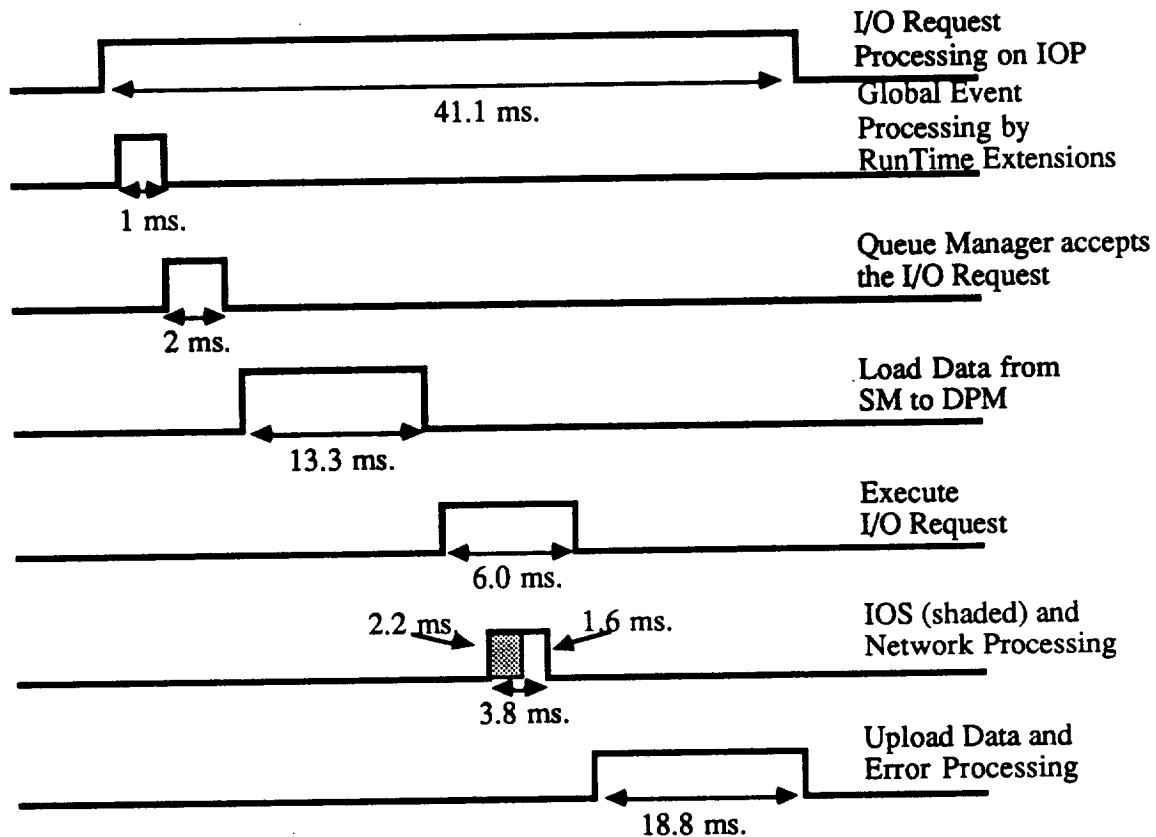


Figure 5-10. I/O Request Processing (10 Hz Task)

Network Bandwidth: 2 Mbits/sec; Node quiet time: 256 ms

H/W: AIPS Engineering Model FTP 68010/7.8 Mhz

S/W: Verdix 5.4

Function

IOP Time

Network FDIR:

| | |
|----------------------------------|---------------|
| Failed Channel | 7 ms. |
| Failed IOS | 111 ms. |
| Failed Link - Disjoint Leaf Node | 181 ms. |
| Failed Leaf Node | 265 ms. |
| Failed Link - Disjoint Branch | 165 - 206 ms. |
| Failed Node - Disjoint Branch | 334 - 360 ms. |

Network Growth:

| | |
|---------------------|--------------|
| Full Diagnostics | 3.55 seconds |
| No Diagnostics | 610 ms. |
| Single Chain Growth | 145 ms. |

Figure 5-11. I/O Redundancy Management Overheads

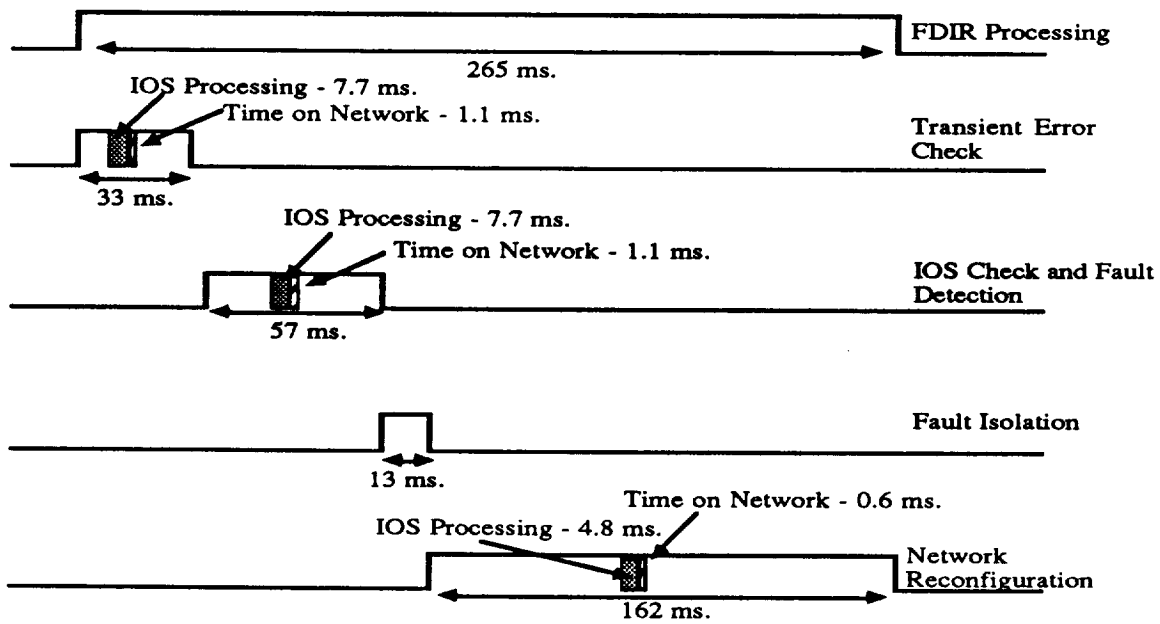


Figure 5-12. I/O Redundancy Management: Failed Leaf Node

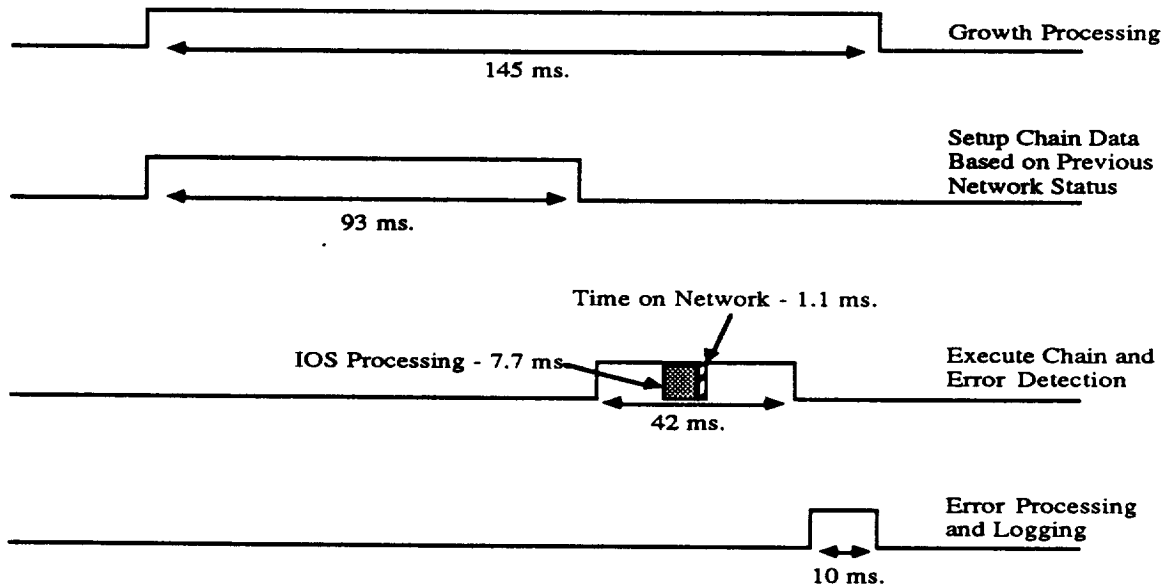


Figure 5-13. I/O Redundancy Management: Single Chain Grow

5.6 INTER-COMPUTER NETWORK EMPIRICAL KNOWLEDGEBASE

5.6.1 IC Network Architecture and H/W and S/W Implementation Technology

A distributed AIPS configuration consists of a number of FTPs which may be physically dispersed throughout a vehicle. These FTPs are linked together by a reliable, damage-tolerant data communication pathway called the Inter-Computer (IC) network. The network consists of a number of full duplex links that are interconnected by circuit switched nodes identical to the I/O nodes and configured into three redundant virtual buses. Each redundant bus is referred to as a network layer. The three layers are totally independent and are not cross-strapped to each other. Each layer contains a circuit-switched node for each processing site; thus every processing site is serviced by three nodes of the IC network. FTPs are designed to receive data on all three layers, but the capability of a FTP to transmit on the network depends on the FTP redundancy level. Triplex FTPs can transmit on all three layers, duplex FTPs on only two of the three layers, and simplex processors on only a single layer. In duplex and triplex FTPs, a given processor can transmit on only one network layer. Thus malicious behavior of a processor can disrupt only one layer.

The distributed AIPS engineering model was designed and implemented in the second development phase of AIPS. Figure 5-14 is a photograph of the AIPS engineering model. The laboratory configuration of the distributed AIPS engineering model, shown in Figure 4-6, consists of four processing sites: three of the processing sites are triplex FTPs; the fourth site is a simplex. Processing site 3 with its 15 node I/O network forms the centralized AIPS configuration, which is a subset of the distributed AIPS configuration. The interfaces between the FTPs and the IC network, shown in Figure 4-6, are called Inter-Computer Interface Sequencers (ICIS). The interfaces between the FTP and the I/O network are called Input/Output Sequencers (IOS).

The IC network and the ICIS are designed in strict accordance with fault-tolerant systems theory. Thus an arbitrary random hardware fault, including Byzantine faults, anywhere in the system can not disrupt communication between triplex FTPs. In other words, the triplex IC network, in conjunction with the ICIS, provides error-masking capability for communication between two triplex computers.

The IC network for the distributed AIPS engineering model configuration consists of three layers of a circuit switched network. Each layer consists of five nodes; each node for each of the four sites and one spare node.

The initial no-fault configurations of the three layers are identical. However, after a link failure in one layer, the virtual bus configuration of that layer will change as the network is reconfigured around the failed link. The other two layers do not have to be reconfigured to make their virtual bus path identical to the third one.

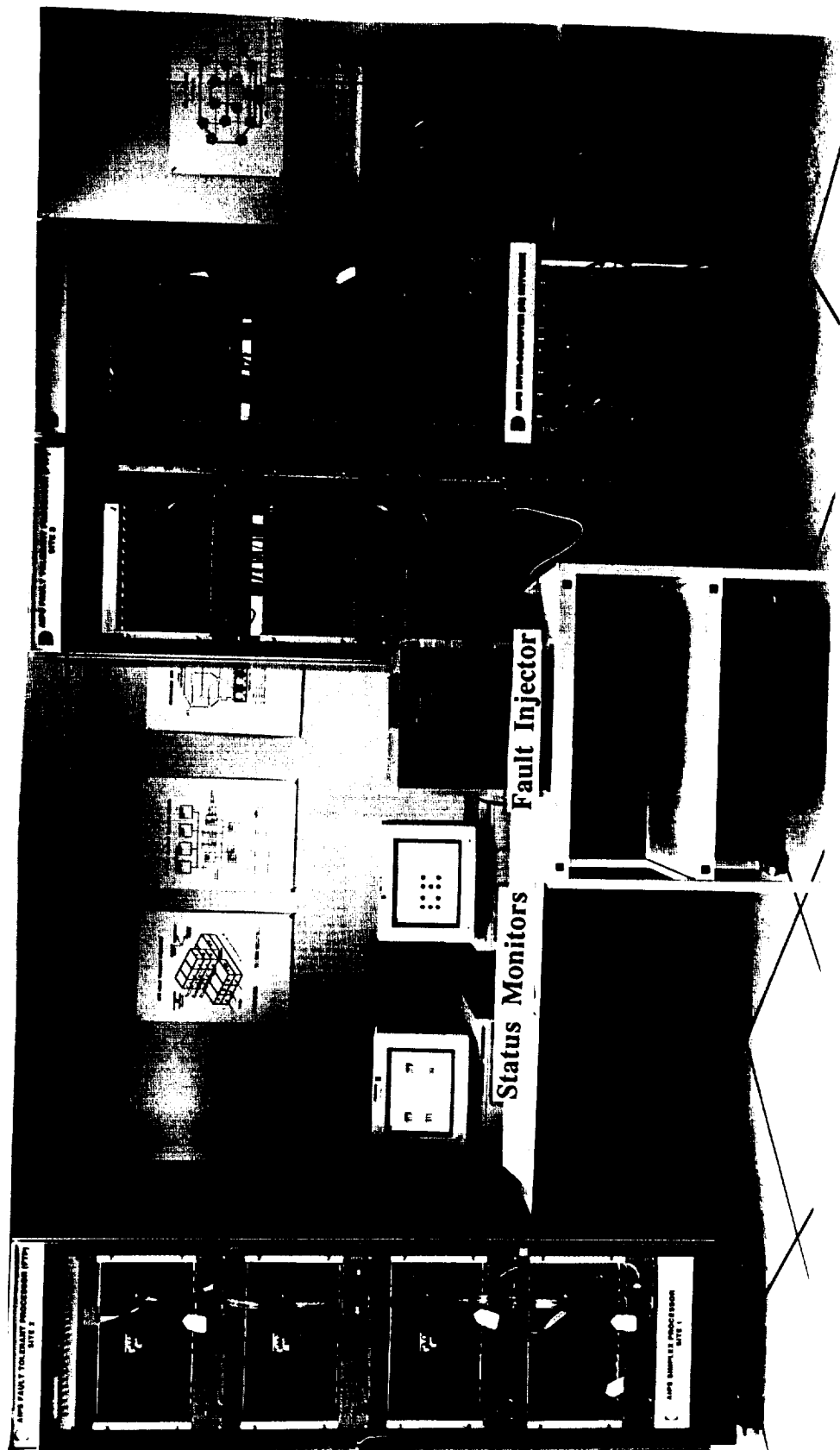


Figure 5-14. AIPS Engineering Model

The fault detection, isolation, and reconfiguration of the IC network are the responsibility of the IC Network Manager software. Nodes keep track of any transmission errors which are protocol related and inform the Manager of these errors when queried by the Manager. This error data can be analyzed by the Network Manager to determine the source of transient faults on the network. The nodes also respond to status queries with the status of the node and the ports. Other than these functions, the nodes are totally passive circuit switching devices. The node has five ports and the common control circuits in a node monitor messages coming in on all five ports whether that port is enabled or not. This procedure is necessary for the initial growth of the network. It is also necessary to monitor all ports so that the Network Manager will be able to respond to certain kinds of failures where the established paths have been disrupted by a malicious failure. The controller decodes the message to determine if it is a valid message and if it is intended for that node. If so the node responds to the message. Messages sent to nodes include requests for status and reconfiguration commands. The Network Manager requests status as an input to its network monitoring task. The reconfiguration messages establish or change the port enable status. Reconfiguration commands are preceded by an encoded node address. Nodes do not respond to messages which are not preceded by valid addresses. Reconfiguration commands are addressed to individual nodes although they are heard by all nodes.

The system software responsible for communication between FTPs is called Inter-Computer Communication Services. A detailed description of the Inter-Computer Communication Services is provided in [4]. The Inter-Computer Communication Services provide two functions: (1) inter-computer user communication services, that is, communication between functions not located in the same FTP, and (2) inter-computer network management .

The IC user communication service provides local and distributed inter-function communication which is transparent to the application user. It provides synchronous and asynchronous communication, performs error detection and source congruency on inputs, and records and reports IC communication errors to IC network layer managers. Inter-Computer communication can be done in either point to point or broadcast mode and is implemented in each FTP.

The IC network manager is responsible for the fault detection, isolation and reconfiguration of the network. The AIPS distributed configuration consists of three identical, independent IC network layers which operate in parallel to dynamically mask faults in a single layer and provide reliable communication. There is one network layer manager for each network layer. However, the three network layer managers do not need to reside in the same FTP. They are responsible for detecting and isolating hardware faults in IC nodes and links and for reconfiguring their respective network layer around any failed elements. The network manager function is transparent to all application users of the network.

The inter-computer communication services has been designed and implemented according to the International Standards Organization (ISO) seven layered model [28].

5.6.2 IC Communication Performance Data

Some performance metrics were gathered in order to determine the overheads for IC communication, using both logic analyzers and the system timer to make the time measurements. All IC performance data was collected on FTP Model 1 using the Verdex 5.5 Ada compiler. These metrics recorded for a typical IC communication between two tasks resident on different sites are presented in Figure 5-15. Twenty samples were taken each time and the numbers were very consistent. The performance was measured from the time the source application task calls the SEND_OUTPUT routine with a message until the time sink application task has message available. The total time was 28.4 milliseconds. The following components of the AIPS distributed engineering model were used:

- 1) two triplex sites (FTP 2 and FTP 3) with 68010 processors with 7.9 MHz clocks
- 2) custom IC Interface Sequencers (ICIS) with 7.9 MHz clocks
- 3) 2 MBit/second IC buses (three)
- 4) 15 custom network nodes
 - a) 68701 processor with a 2MHz clock
- 5) Verdex 5.5 compiler/RTS
- 6) Message length was 64 bytes

As indicated by Figure 5-15, the time on the network does not cause extensive performance penalties. Instead, the performance penalties are caused by the processor, compiler and data exchange hardware. A state-of-the-art processor and a mature Ada compiler should increase performance significantly. The impact of these technology insertions are discussed in the following subsection.

5.7 Impact of Advanced Technology Insertions

The design and implementation of the AIPS hardware and software building blocks was begun in 1983. Therefore, the Ada compiler used for most of the development and performance metrics was an immature compiler. In the area of hardware, the microprocessor used for the AIPS FTPs was a Motorola 68010 with a 7.9 MHz clock. This hardware was used for both the I/O network and IC communication performance evaluation. In order to project the 1992 performance characteristics for AIPS/ALS, the impact of software and hardware advanced technology insertions was studied. Software technology controls basic parameters such as the Ada Rendezvous time which depends among other things on the Ada compiler technology. A number of compilers were benchmarked to determine the impact of advanced software technology on performance.

| <u>Location</u> | <u>Function</u> | <u>Overhead(Verdix 5.5)</u> |
|------------------------|---|-------------------------------------|
| Source FTP | SEND_OUTPUT (Add message to output queue, set event) | 3.7 ms |
| Source FTP | Time between event and MSR | 1.4 ms |
| Source FTP | MSR | 5.7 ms |
| IC Net | Time for ICIS and IC Network transmission | .4 ms |
| Sink FTP | Ave time between polling for msg | 2.5 ms |
| Sink FTP | ICIS RM (make msg congruent, check for errors) | 9.7 ms |
| Sink FTP | Time to do context switch | .9 ms |
| Sink FTP | MSR (Time between start and when its in buffer for user) | 3.2 ms |
| Sink FTP | GET_INPUT (Remove message from buffer and pass to user) | <u>.8 ms</u> |
| | Total Time | 28.4 ms. |

Figure 5-15. IC Communication Overhead

Parameters that are a function of the hardware technology are the raw CPU throughput, mean time between failures (MTBF) of CPU, memory, and raw bandwidth of I/O and IC networks. A number of microprocessors were studied to determine the impact of advanced hardware technology insertion on system performance.

Section 5.7.1 discusses the performance improvement gained from a mature Ada compiler and Section 5.7.2 discusses the performance improvement gained from state-of-the-art microprocessors.

5.7.1 Software Technology Insertion - Compilers

Dhrystone and Whetstone benchmarks were used in order to determine the performance improvement of the FTP with several Ada compilers since the AIPS system

software is written in Ada. The purpose of using these benchmark tests with several Ada compilers was to gain performance information about the behavior and influence of compilers for the AIPS knowledgebase. The Dhrystone benchmark results to date are shown in Figure 5-16. The Dhrystone benchmark was conceived by R. Weicker for typical system software code. No floating operations are performed, but it does make use of record and pointer data types. The Dhrystone is also a useful measure of integer performance. The compilers used for the measurements were Verdix Ada 5.4, Verdix Ada 5.5, Verdix Ada 5.7, Green Hill C, and XD Ada (Beta 2). These benchmarks were done on two engineering models of AIPS. The results of the benchmarks are presented in Figure 5-16. The checks on/checks off column refers to the run time checks made by Ada compilers for out-of-range data.

Changing the microprocessor from a 7.8 MHz M68010 (FTP Model 1) to a 14.5 MHz M68020 (FTP Model 2) improved the throughput performance by 560 % to 590 %. The advanced hardware technology insertions will be discussed in more detail in the next section. FTP Model 1 was used to collect all performance data presented in Sections 5.5.2, 5.5.3, and 5.6.2 since the I/O and IC network interfaces were not designed and fabricated for FTP Model 2.

The performance improvement gained from the series of Verdix compilers as they matured is 18%. The compiler performance improvement gained between Verdix 5.4 and the XD Ada compiler is 70%.

| <u>Computer</u> | <u>Compiler</u> | <u>Dhrystone/Sec.</u> Cks. On/Cks.Off |
|--|-----------------|--|
| FTP Model 1 (68010 7.8MHz clock) | Verdix Ada 5.4 | 384 |
| FTP Model 1 (68010 7.8MHz clock) | Verdix Ada 5.5 | 413 |
| FTP Model 2 (68020/68882/14.5MHz clock) | Verdix Ada 5.4 | 2166 |
| FTP Model 2 (68020/68882/14.5MHz clock) | Verdix Ada 5.5 | 2452 |
| FTP Model 2 (68020/68882/14.5MHz clock) | Verdix Ada 5.7 | 2552 |
| FTP Model 2 (68020/68882/14.5MHz clock) | Green Hill C | /3179 |
| FTP Model 2 (68020/68882/14.5MHz clock) | XD ADA (Beta2) | 3670 /5358 |

Figure 5-16. Dhrystone (Compiler) Benchmark Results

The Whetstone benchmark results to date are shown in Figure 5-17. It provides a good metric of a microprocessor's performance while executing typical scientific software. Since the Whetstone is a measurement of floating point intensive computation, this benchmark was only done on FTP Model 2, which has a 68882 floating point co-processor. (FTP Model 1 does not have a floating point co-processor.) Ada compilers do not fully support the floating point 68882 co-processor; therefore floating point co-processor routines were added to Verdex 5.4, 5.5, and XD Ada (Beta 2). These Draper additions are indicated by the '+' in Figure 5-17.

| <u>Computer</u> | <u>Compiler</u> | <u>K Whetstones/Sec.</u> |
|--|-----------------|--------------------------|
| FTP Model 2 (68020/68882/14.5MHz clock) | Verdex Ada 5.4+ | 493 |
| FTP Model 2 (68020/68882/14.5MHz clock) | Verdex Ada 5.5+ | 865 |
| FTP Model 2 (68020/68882/14.5MHz clock) | XD Ada Beta 2 + | 929 |

Figure 5-17. Whetstone (Compiler) Benchmark Results

The compiler performance improvement gained between Verdex 5.4 and the XD Ada compiler is 88%.

5.7.2 Hardware Technology Insertion - Microprocessors

The hardware technology survey [7] shows that the microprocessor performance can be expected to increase dramatically over the next several years. In particular, the introduction of RISC (Reduced Instruction Set Computers) based architectures, 32-bit data and address paths, and higher clock frequencies will contribute significantly to the increased throughput performance. It is expected that in the ALS Preliminary Design Review time-frame (circa 1992), the JIAWG (Joint Integrated Avionics Working Group) microprocessors such as the MIPS R3000 and the Intel 80960 32-bit RISC machines will be operational in 40 MHz clock rates for flight system applications. The details of the AIPS FTP that utilizes such a microprocessor are provided in the AIPS for ALS Architecture Synthesis Report [8].

The technology survey also collected a number of throughput benchmarks results for a number of different microprocessors. Based on these benchmarks as well as similar benchmark data collected on the AIPS FTPs, we extrapolate conservatively that the AIPS/ALS FTP, programmed in Ada using the XD Ada compiler, can be expected to deliver 40K Dhrystones per second and 18 Double Precision Mega-Whetstones per second.

The AIPS/ALS Fault Tolerant Processor is projected to have a throughput performance that is 100 times better than FTP Model 1 (7.8 MHz M68010 based FTP) for system software and integer code. For floating point operations, the AIPS/ALS FTP will be about 20 times faster than FTP Model 2 (14.5 MHz M68020/68882 based FTP). The impact of both software and hardware technology is summarized in the following section.

5.8 Conclusions

A set of architectural parameters have been defined for the AIPS distributed fault tolerant computer architecture that a system designer can choose to meet the performance and reliability requirements for a specific application. A set of performance and reliability metrics have also been defined that can be used to evaluate and compare computer architectures for real time mission- and safety- critical aerospace applications. The performability data measured on two AIPS engineering models was also presented. The major conclusions based on this empirical data are as follows.

The Redundancy Management overheads for the Motorola 68020-based AIPS Fault Tolerant Processor consume only 2% of the processor throughput under nominal no-fault conditions.

The Ada operating system overheads consume approximately 6% of the processor throughput with an average mix of 4 timer based dispatches and 1 local event dispatch per frame for a 68020-based FTP. This has decreased with the more mature XD Ada compiler and it is likely to decrease even further.

The synchronization of redundant processors in the FTP slows down the FTP processors by about 5%. Thus about 87% of the raw processor throughput is available for the applications tasks after accounting for various overheads. This makes the AIPS FTP extremely efficient and competitive even with simplex processors.

The throughput of the Motorola 68020-based engineering model FTP, programmed in XD Ada compiler, was benchmarked at about 5000 Dhrystones/sec per processor or about 10,000 Dhrystones/sec for the AIPS FTP which consists of 2 processors per channel. This throughput is expected to be about 80,000 Dhrystones/sec for the AIPS/ALS FTP which is expected to use a state-of-the-art 40 MHz 32-bit RISC microprocessor. The throughput performance of the AIPS/ALS FTP is expected to be about 100 times better than the original AIPS engineering model FTP which used the Motorola 68010 microprocessors.

The dominating factors in the I/O Redundancy Management and overhead are found to be the processor, compiler and FTP data exchange and not the I/O network or node speed. In the AIPS FTP Model 1, redundant networks are necessary to recover from a

network fault while continuing to access sensors in real time. A state-of-the-art microprocessor and a mature Ada compiler would increase the performance by a factor of 100 which will allow an I/O network to recover from a fault in real time. For example, a single-chain total regrowth of a 15-node I/O network will take less than 2 msecs.

The dominating factors in the Inter-Computer Communications overhead are found to be the processor, compiler and FTP data exchange and not the IC network or node speed. A state-of-the-art microprocessor would increase the performance by a factor of 100 which will allow inter-computer communication in real time. For example, the average time to send a message over the inter-computer network, from a task in one FTP to another task on a different FTP, will be less than 3 msecs.

Future plans include measurement of intercomputer Redundancy Management overheads, probability distribution functions for fault detection, isolation and reconfiguration, and other performance and reliability metrics mentioned in this section.

6.0 SUMMARY AND CONCLUSIONS

This report has defined a design methodology that can be used to produce validated fault tolerant distributed computer system architectures, suitable for a broad range of advanced aerospace applications, in a cost effective manner. The design for validation methodology uses a substantial body of knowledge that we collectively call the architecture design and validation knowledgebase. Such a knowledgebase has been created for the AIPS architecture over the past few years. A part of that knowledgebase has been presented in this report. Other parts of the AIPS knowledgebase can be found in other related NASA reports [1-9].

The total body of knowledge necessary to design and validate distributed fault tolerant computer system architectures is vast. Although significant progress has been made in acquiring, organizing, and documenting this knowledgebase for the AIPS architecture, much work remains to be done in making it complete. This section summarizes the salient results of the present study and recommends steps that are necessary to complete the acquisition and organization of the remaining parts of the knowledgebase.

6.1 Architecture Knowledgebase

The relationships among AIPS mission requirements, architectural attributes, rules, and guidelines have been constructed and expressed through the use of a Directed Acyclic Graph (DAG) conceptualization. The relationships depicted in the directed graphs allow a designer unacquainted with the details of fault tolerance technology to understand key issues regarding the organization and operation of AIPS. They also provide a framework which permits traceability of the AIPS mission requirements to the AIPS design specifications, a traceability which it is believed will enhance the AIPS validatability. The approach also provides a pedagogy for the AIPS fault tolerance theory, and explains the rationale behind the AIPS approach to fault tolerance.

This is the first time a transitive relationship between mission requirements, architectural attributes, rules, and guidelines using DAG has been attempted for a fault tolerant distributed computer architecture. Naturally, the overall organization of this knowledgebase is less than optimal. Having gone through the exercise of constructing these directed acyclic graphs once, it is much more clear how this complex knowledgebase could be better organized. Section 2.7.2 provides detailed recommendations for future work in this area. Furthermore, steps that are necessary to complete the architecture knowledgebase are also contained in this section.

6.2 Performability Knowledgebase

The performance and reliability, or performability, of the AIPS building blocks has been characterized using analytical models and via empirical measurements on the AIPS

engineering models. The analytical models, though general in nature, were solved using the parameters for the Advanced Launch System mission scenario.

The reliability analysis results indicate that a quadruply redundant AIPS Fault Tolerant Processor and a quadruply redundant Inter-Computer network will meet the ALS availability and reliability requirements discussed in Section 4 without any need for reconfiguration. Specifically, these requirements include a 95 per cent availability of launch vehicle avionics after a 1 week unattended operation on the launch pad and a probability of failure that is less than 10^{-5} for the boost phase and on-orbit operations.

The performance of the AIPS building blocks was modeled in the context of the ALS avionics system performance requirements. While much progress was made in extracting ALS requirements from Martin Marietta, additional work needs to be done to obtain avionics requirements at the task-level granularity. Also, a mutually meaningful definition of I/O and inter-function communication requirements needs to be determined and the relevant parameters quantified.

In the modeling effort, some preliminary ADAS models of the FTP were constructed. Additional lower-level ADAS models of the detailed workings of the FTP, IOS, and I/O network need to be constructed. The model also needs to be expanded to include multiple FTPs communicating over the Inter-Computer network. The task granularity requirements and the multi-FTP ADAS models can then be used for estimation of the AIPS performance. In this context, the manual process of allocating ALS functions to multiple FTPs and to the computational and the I/O processors within an FTP also needs to be automated so that a more optimal task allocation can be achieved.

Significant performance results were also collected on the AIPS engineering models. These measurements validated the claims of very efficient implementation of fault tolerance on AIPS. For example, the redundancy consumes less than 2 percent of processor throughput on Motorola 68020-based FTP. And the synchronization of redundant processors in the FTP slows down the processors by less than 5 per cent. The Ada realtime operating system consumes an additional 6 per cent of processor throughput assuming an average mix of 4 timer-based task dispatches and 1 local event-based dispatch per frame. Thus, about 87% of raw processor throughput is available for applications tasks in the FTP. This makes the AIPS FTP extremely efficient and competitive even with simplex processors.

The throughput of the Motorola 68020-based engineering model FTP, programmed in XD Ada compiler, was benchmarked at about 5000 Dhrystones/sec per processor or about 10,000 Dhrystones/sec for the AIPS FTP which consists of 2 processors per channel. This throughput is expected to be about 80,000 Dhrystones/sec for the AIPS/ALS FTP which is expected to use a state-of-the-art 40 MHz 32-bit RISC microprocessor. The throughput performance of the AIPS/ALS FTP is expected to be about 100 times better

than the original AIPS engineering model FTP which used the Motorola 68010 microprocessors.

The dominating factors in the I/O Redundancy Management and overhead are found to be the processor, compiler and FTP data exchange and not the I/O network or node speed. In the AIPS FTP Model 1, redundant networks are necessary to recover from a network fault while continuing to access sensors in real time. A state-of-the-art microprocessor and a mature Ada compiler would increase the performance by a factor of 100 which will allow an I/O network to recover from a fault in real time. For example, a single-chain total regrowth of a 15-node I/O network will take less than 2 msecs.

The dominating factors in the Inter-Computer Communications overhead are found to be the processor, compiler and FTP data exchange and not the IC network or node speed. A state-of-the-art microprocessor would increase the performance by a factor of 100 which will allow inter-computer communication in real time. For example, the average time to send a message over the inter-computer network, from a task in one FTP to another task on a different FTP, will be less than 3 msecs.

Future plans include measurement of intercomputer Redundancy Management overheads, probability distribution functions for fault detection, isolation and reconfiguration, and other performance and reliability metrics mentioned in this section.

6.3 Formal Proofs

This project comprises a first step in the formal specification and verification of the interactive consistency function for ultrareliable digital computing systems. The phases of a complete formal specification and verification effort are to (1) formally specify the concept of interactive consistency, (2) develop and specify a detailed finite-state machine model, (3) develop a circuit-level implementation, and (4) develop mathematical mappings between each level of specification.

The current project, which comprised only phase (1) of the overall effort, was carried out without unreasonable difficulty by non-experts in the field of formal verification. No major problems were found or foreseen in this phase which would make the completion of phases (2), (3), or (4) unduly difficult. Successful completion of these phases would result in the availability of a verified and valuable building block for the construction of ultrareliable digital systems.

7.0 REFERENCES

1. Burkhardt, L., L. Alger, R. Whittredge, and P. Stasiowski, "Advanced Information Processing System: Local System Services", NASA Contractor Report 181767, April, 1989.
2. Masotto, T., and L. Alger, "Advanced Information Processing System: Input/Output System Services", NASA Contractor Report 181874, August 1989.
3. Nagle, G., L. Alger and A. Kemp, "Advanced Information Processing System: Input/Output Network Management Software", NASA Contractor Report 181678, May 1988.
4. Burkhardt, L., T. Masotto, J. Terry Sims, R. Whittredge, and L. Alger, "Advanced Information Processing System: Inter-Computer Communication Services", NASA Contractor Report 187556, September 1991.
5. Lala, J.H. and S.J. Adams, "Inter-Computer Communication Architecture for a Mixed Redundancy Distributed System", Journal of Guidance, Control, and Dynamics, Vol. 12, No. 4, July-August 1989.
6. Sims, T., "Advanced Information Processing System Inter-Computer Network Data Source Congruency Algorithm Simulation", CSDL Memo on IC Redundancy Management, Memo No. AIPS-87-11, April, 1987.
7. Cole, R., "Advanced Information Processing System for Advanced Launch System: Hardware Technology Survey and Projections", NASA Contractor Report 187555, September 1991.
8. Lala, J.H., R. Harper, K. Jaskowiak, L. Alger, G. Rosch, and A. Schor, "Advanced Information Processing System for Advanced Launch System: Avionics Architecture Synthesis", NASA Contractor Report 187554, September 1991.
9. Rushby, J., F. von Henke, and S. Owre, "An Introduction to Formal Specification and Verification Using EHDM", Computer Science Laboratory, SRI International, Technical Report SRI-CSL-91-2, Menlo Park, CA, February 1991.
10. Lala, J.H., L.S. Alger, R.J. Gauthier, and M.J. Dzwonczyk, "A Fault Tolerant Processor Architecture to Meet Rigorous Failure Requirements," 7th AIAA -IEEE Digital Avionics Systems Conference, Fort Worth, TX, October 1986, CSDL-P-2705.

11. Gauthier, R.J., J.H. Lala, and J.J. Deyst, Jr., "Advanced Avionics Technology Demonstration: Alternative Fault-Tolerant Architectures Study", CSDL-R-1949, Prepared for Naval Air Development Center, Wainwright, PA, April 1987.
12. Lala, J.H., "An Advanced Information Processing System", 6th AIAA-IEEE Digital Avionics Systems Conference, Baltimore, MD, Dec. 1984, CSDL-P-1952.
13. Alger, L. and J. Lala, "Performance Evaluation of a Real-Time Fault Tolerant Distributed System", 23rd Hawaii International Conference of System Sciences, Kailua-Kona, Hawaii, January 1990.
14. "Advanced Information Processing System Plan," CSDL-R-2032, October 1987.
15. "Advanced Information Processing System (AIPS) Proof-of-Concept System: Fault Tolerant Processor Requirements/Design Specification," CSDL-AIPS-84-161, June 1985.
16. "AIPS System Requirements," CSDL Report No. AIPS-83-50, August 1983.
17. "AIPS System Specification," CSDL Report No. CSDL-C-5709, May 1984.
18. "AIPS POC System I/O Network System Services Functional Requirements," CSDL-AIPS-84-138, October 1984.
19. "AIPS POC System FTP Requirements/Design Specifications," CSDL-AIPS-84-161, June 1985.
20. "AIPS POC System Network Node Requirements/Design Specifications," CSDL-84-162, July 1985.
21. "AIPS POC System Functional Design of Communication Services," CSDL-AIPS-85-04, May 1985.
22. "AIPS POC System Functional Design (Rev. 1), CSDL-AIPS-85-81, November 1985.
23. "AIPS POC System Phase II Configuration Management Plan," CSDL-AIPS-85-105, November 1985.
24. "AIPS POC System Software Requirement Specifications Builds 1 and 2," CSDL-AIPS-85-113, June 1985.

25. "AIPS POC System Software Requirement Specifications Build 3," CSDL-AIPS-85-115, July 1985.
26. "AIPS POC System AIPS I&E Facility: Functional Reports and Description," CSDL-AIPS-85-158, August 1985.
27. "AIPS POC System Software Requirements Specifications Build 4," CSDL-AIPS-85-246, January 1986.
28. Martin, J. Distributed Processing Software and Network Strategy, Savant Research Studies, Lancashire, England, October 1979.
29. Hopkins, A.L., Jr., J.H. Lala, and T.B. Smith, III, "The Evolution of Fault Tolerant Computing at the Charles Stark Draper Laboratory, 1955-85", Dependable Computing and Fault-Tolerant Systems, Vol. 1: The Evolution of Fault-Tolerant Computing, Eds: Avizienis, Kopetz, Laprie, Springer Verlaag, pages 121-140, May 1987.
30. Cullyer, W.J., Pygott, C.H., "Hardware Proofs using LCF-LSM and ELLA," Royal Signals and Radar Establishment Memorandum No. 3832, September 1985.
31. Lamport, L., Shostak, R., Pease, M., "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982.
32. Dzwonczyk, M. and H. Stone, "A Fault-Tolerant Avionics Suite for An Entry Research Vehicle", 8th AIAA/IEEE Digital Avionics Systems Conference, San Jose, CA, October 1988, CSDL-P-2810.
33. Lala, J.H., "A Byzantine Resilient Fault-Tolerant Computer for Nuclear Power Plant Applications", 16th International Symposium on Fault Tolerant Computing, Vienna, Austria, July, 1986.



Report Documentation Page

| | | | |
|---|--|---|-----------|
| 1. Report No. NASA CR-187544 | 2. Government Accession No. | 3. Recipient's Catalog No. | |
| 4. Title and Subtitle Advanced Information Processing System: Design and Validation Knowledge Base | | 5. Report Date September 1991 | |
| | | 6. Performing Organization Code | |
| 7. Author(s) Richard E. Harper, Linda S. Alger, and Jaynarayan H. Lala | | 8. Performing Organization Report No. | |
| | | 10. Work Unit No. 506-46-21-56 | |
| 9. Performing Organization Name and Address The Charles Stark Draper Laboratory, Inc. Cambridge, MA 02139 | | 11. Contract or Grant No. NAS1-18565 | |
| | | 13. Type of Report and Period Covered Contractor Report | |
| 12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225 | | 14. Sponsoring Agency Code | |
| | | | |
| 15. Supplementary Notes Langley Technical Monitor: Felix L. Pitts Final Report | | | |
| 16. Abstract The overall objective of the Advanced Information Processing System (AIPS) program is to develop the knowledge base which will allow achievement of validated fault-tolerant distributed computer system architectures, suitable for a broad range of applications, including those which have a failure probability requirement as low as 10^{-9} at 10 hours. The specific quantitative and qualitative design objectives of AIPS are discussed later in this report. The purpose of this report is to provide a comprehensive detailed documentation of the knowledge base which forms the foundation of AIPS. | | | |
| 17. Key Words (Suggested by Author(s)) Fault-Tolerant Digital Computers Validation Knowledge Base Fault Tolerance Distributed Processing | | 18. Distribution Statement Unclassified - Unlimited Subject Category 62 | |
| 19. Security Classif. (of this report) Unclassified | 20. Security Classif. (of this page) Unclassified | 21. No. of pages 194 | 22. Price |